VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Detekcia lokálnych komunít v komplexných sieťach

Detection of Local Communities in Complex Networks

2014

Bc. Martin Gajdičiar

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Martin Gajdičiar**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Detekce lokálních komunit v komplexních sítích
Detection of Local Communities in Complex Networks

Zásady pro vypracování:

Jednou z možností detekce komunit v komplexních sítích je použití algoritmů pro hledání lokálních komunit budovaných zdola zpravidla od počátečního vrcholu či hrany. Existuje mnoho různých přístupů, z nichž některé naleznou takzvané překrývající se komunity. Cílem práce je výběr a implementace takovýchto algoritmů, provedení experimentů nad reálnými daty a interpretace výsledků.
1. Seznamte se s problematikou detekce komunit v komplexních sítích.
2. Prostudujte existující přístupy k detekci lokálních překrývajících se komunit.
3. Vyberte vhodné metody a vhodná testovací data.
4. Vybrané metody naimplementujte, proveďte experimenty a jejich výsledky vyhodnoťte.

Seznam doporučené odborné literatury:

[1] M. E. J. Newman, Networks: An Introduction, Oxford University Press (2010), ISBN-10: 0199206651.
[3] G. Palla, I. Derényi, I. Farkas, and T. Vicsek: Uncovering the overlapping community structure of complex networks in nature and society. Nature 435, 814-818 (2005).
[2] Dle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **RNDr. Eliška Ochodková, Ph.D.**

Datum zadání: 01.09.2013
Datum odevzdání: 07.05.2014

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

V Ostrave 7. mája 2014 .....

Na tomto mieste by som sa rád poďakoval vedúcej práce RNDr. Eliške Ochodkovej Ph.D., za jej cenné odborné rady, trpezlivosť, poskytnuté študijné materiály a čas strávený pri konzultáciách.

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Poděkování

Tato práce byla vypracována s podporou projektu Rozvoj lidských zdrojů ve výzkumu a vývoji moderních soft computingových metod a jejich praktického využití, reg. č. CZ.1.07/2.3.00/20.0072 podpořeného Operačním programem Vzdělávání pro konkurenceschopnost, financovaného ze strukturálních fondů EU a státního rozpočtu ČR.

Abstrakt

Táto práca sa zaoberá problematikou vyhľadávania lokálnych, prekrývajúcich sa komunít v komplexných sieťach. Analýza komplexných sietí sa v poslednom desaťročí stala, vďaka rozmachu sociálnych sietí a dostupnosti dát na WWW, veľmi populárnou disciplínou. Veľké úsilie bolo vyvinuté práve v oblasti detekcie komunít v komplexných sieťach. Prístupov k detekcii komunít existuje niekoľko, no môžeme ich rozdeliť do dvoch hlavných vetví a to na detekciu lokálnych a na detekciu globálnych komunít. Tie sa ďalej delia na základe toho, či umožňujú vrcholom patriť do viacerých komunít, alebo nie. V práci popíšeme špecifiká jednotlivých prístupov, pričom sa budeme zaoberať hlavne lokálnymi metódami. Podrobne rozoberieme dve metódy, ktorých variácie taktiež naiplementujeme a to konkrétne *Clique percolation method (CPM)* a *Connected iterative scan (CIS)*. Nakoniec budú v práci uvedené experimenty, porovnávajúce implementované metódy a interpretácia ich výsledkov.

Kľúčové slová: komplexné siete, graf, komunita, detekcia komunít, lokálne komunity, prekrývajúce sa komunity, CIS, CPM, NCIS, GCIS

Abstract

This thesis deals with problems of local overlapping community detection in complex networks. During the last ten years complex network analysis has become very popular discipline thanks to the growth of social networks and data availability on the WWW. Big effort was involved especially in the field of community detection in complex networks. There are few different approaches to the community detection, but they can be divided into two main branches. Either we can detect global or local communities. These are further divided depending on fact if they allow vertices to belong to multiple communities or not. We will describe the differences and specifics of each method, but we will focus mainly on local methods. Two methods which variations were implemented will be discussed in detail. These are *Clique percolation method(CPM)* and *Connected iterative scan (CIS)*. In the end of this thesis we will discuss performed experiments comparing implemented methods and their results.

Keywords: complex networks, graph, community, community detection, local communities, overlapping communities, CIS, CPM, NCIS, GCIS

Zoznam použitých zkratiek a symbolov

IS	–	Iterative Scan
RaRe	–	Rank Removal
CIS	–	Connected Iterative Scan
CPM	–	Clique Percolation Method
NCIS	–	Neighbors Connected Iterative Scan
GCIS	–	Greedy Connected Iterative Scan
LFM	–	Lancichinetti-Fortunato Method
EAGLE	–	Agglomerative hierarchical clustering based on maximal clique
VDM	–	Local community detection based on vertex dependency
MONC	–	Merging of Overlapping Natural Communities
GCE	–	Greedy Clique Expansion
BFS	–	Breadth First Search
WWW	–	World Wide Web
HSB	–	Hue, Saturation, Brightness color model
GEXF	–	Graph Exchange XML Format
CSV	–	Comma Separated Values
TXT	–	Text file
SVG	–	Scalable Vector Graphics
GB	–	Gigabyte
RAM	–	Random Access Memory
PC	–	Personal Computer
Sek.	–	Sekcia
Obr.	–	Obrázok
Str.	–	Strana

Obsah

Zoznam tabuliek

Zoznam obrázkov

Zoznam výpisov zdrojového kódu

1 Úvod

V poslednom desaťročí sa vďaka dostupnosti dát na internete a nárastu výpočtového výkonu počítačov stala veľmi obľúbenou problematikou analýza komplexných sietí. Príkladov komplexných sietí, resp. systémov, ktoré môžeme pomocou nich reprezentovať, nájdeme na internete, ale aj v reálnom svete mnoho. Jedná sa napr. o siete spolupráce, citačné siete, dopravné siete, sociálne siete, WWW, elektrické rozvodné siete, nervové siete v biologických systémoch a mnohé iné.

Komplexné siete matematicky reprezentujeme pomocou grafov, kde jedinci, alebo objekty siete predstavujú vrcholy a nejaký definovaný typ interakcie medzi nimi predstavuje hrany. Oproti jednoduchým grafom ako sú mriežky, alebo náhodné grafy, sa líšia v tom, že majú netriviálne topologické vlastnosti. Medzi ne patrí okrem iného to, že obsahujú *komunity*.

Už z uvedených príkladov komplexných sietí je jasné, že svoje pomenovanie nedostali náhodou. Jedná sa naozaj o veľmi rozsiahle štruktúry, často s miliónmi vrcholov a hrán. Práve táto komplexnosť veľmi sťažuje ich analýzu. Vedci sa zaoberajú skúmaním rôznych vlastností sietí vrátane topológie, ktorá je veľmi dôležitá, pretože štruktúra ovplyvňuje procesy v sieti. Napr. topológia sociálnych sietí ovplyvňuje rýchlosť šírenia informácií, či chorôb a môžeme z nej vyčítať, ktorí jednotlivci majú v sieti najväčší vplyv. Ak teda pochopíme štruktúru siete, môžeme odhadnúť jej správanie.

Pod analýzu štruktúry sietí spadá aj vyhľadávanie komunít, ktorému sa budeme v tejto práci venovať. Identifikácia komunít je netriviálna úloha, ktorá sa stala predmetom intenzívneho výskumu posledných rokov. Komunitu si môžeme v jednoduchosti predstaviť, ako skupinu objektov, ktoré medzi sebou komunikujú viac, ako so svojim okolím. Detekcia komunít je v mnohých typoch sietí veľmi užitočná. Napr. D.M. Wilkinson a B.A. Huberman v [28] analyzovali sieť výskytov génov za účelom identifikácie skupín príbuzných génov. Úspešne ukázali, že gény patriace do rovnakej skupiny sú si aj funkčne podobné. Pomocou zoskupovania podobných génov môžeme napr. určiť ich vzťah k určitej chorobe.

Metód na detekciu komunít existuje mnoho, no môžeme ich rozdeliť do niekoľkých skupín. Hlavné delenie je na *lokálne* a *globálne* metódy. Ďalej sa metódy delia na také, ktoré umožňujú vrcholom patriť do viac ako jednej komunity súčasne, alebo naopak takých, ktoré priraďujú vrchol vždy do jednej komunity. Rozdiely medzi jednotlivými prístupmi budú popísané v kapitole 3.

Práca si kladie za cieľ analyzovať dostupné lokálne metódy slúžiace na detekciu prekrývajúcich sa komunít, vybrať a implementovať vhodné metódy a porovnať ich výsledky. V jednotlivých kapitolách čitateľa postupne uvedieme do problematiky komplexných sietí a ich reprezentácie 2, ďalej sa budeme bližšie venovať komunitám v takýchto sieťach 3 spolu s popisom prístupov k ich detekcii. V kapitole 3 rovnako popíšeme niekoľko algoritmov z každej kategórie detekcie, ale dôraz budeme klásť hlavne na lokálne metódy. Potom sa budeme podrobne venovať popisu metód *Clique percolation method* a *Connected iterative scan* 4, ktoré sme ako súčasť tejto práce implementovali. Nakoniec budú uvedené experimenty nad rôznymi sieťami spolu s analýzou ich výsledkov 6.

2 Komplexné siete a ich reprezentácia

V úvode sme spomenuli niekoľko príkladov dát, resp. systémov reprezentovaných pomocou sietí. V tejto kapitole uvedieme niektoré základné vlastnosti komplexných sietí. Podrobnejšie rozoberieme typy komplexných sietí a stručne uvedieme problémy týkajúce sa ich analýzy. Keďže **siete sú grafy**, tak sa budeme najskôr venovať základným pojmom z teórie grafov, ktoré sú nutné k ďalšiemu pochopeniu tejto problematiky.

2.1 Základy teórie grafov

Počiatok teórie grafov datujeme do roku 1736, kedy matematik L. Euler zistil, že problém „7 mostov mesta Kaliningrad“ nemá riešenie [31]. Odvtedy sme sa toho o grafoch a ich matematických vlastnostiach naučili veľa.

2.1.1 Graf

Pojmom graf, budeme v tejto práci označovať grafický spôsob vyjadrenia vzťahu medzi objektmi. Objekty sú v grafe reprezentované *vrcholmi*. V rôznych oblastiach sa používajú rôzne pojmy, ktoré označujú vrchol. Napr. v sociológii sa hovorí o *aktéroch*, či *jedincoch*, ktorí sú prepojení *väzbami*, resp. *vzťahmi*. My budeme v tejto práci používať označenie *vrchol* a *hrana*, kde hrana reprezentuje vzťah medzi dvoma vrcholmi.

Sila teórie grafov spočíva hlavne v stručnej a prehľadnej interpretácii problému, kedy abstrahujeme od nedôležitých symbolov a sústredíme sa na štruktúru, ktorá je v problému obsiahnutá [44].

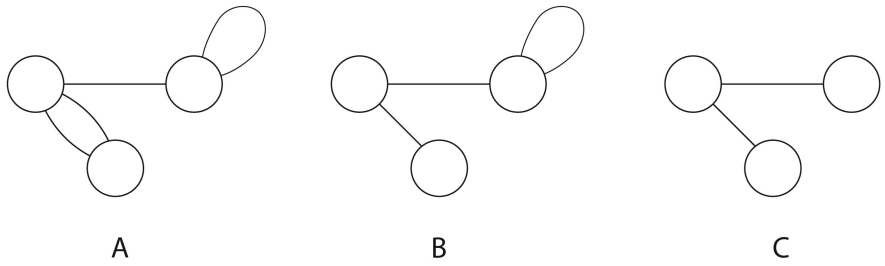
Grafy znázorňujeme pomocou diagramov. Pri kreslení grafu sa väčšinou vrcholy reprezentujú kolieskom a hrany kreslíme priamymi, alebo zaoblenými čiarami. Hrana vždy začína a končí v nejakom vrchole. Väčšinou sú koncové vrcholy rôzne, ale môže to byť aj jeden vrchol. Takejto hrane hovoríme *slučka*. Dva vrcholy môžu byť spojené viacerými hranami, v takom prípade ich označujeme ako *násobné*. Graf, ktorý obsahuje násobné hrany nazývame *multigraf*. Graf bez násobných hrán sa nazýva *prostý graf* – môže však obsahovať slučky. Prostý graf bez slučiek je *obyčajný graf* [45]. Ukážky jednotlivých typov grafov sú na obr. 1.

Hrany v grafe môžu byť :

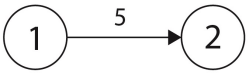
- *Neorientované* – reprezentujú symetrické vzťahy medzi vrcholmi. Napr. ak chceme označiť vzťah priateľstva.
- *Orientované* – reprezentujú jednosmerné, teda nesymetrické vzťahy medzi vrcholmi. Orientácia hrany sa označuje šípkou na jednom jej konci. Napr. ak chceme znázorniť vzťah, že vrchol je rodičom iného vrcholu.

a ďalej :

- *Ohodnotené* – hrany, ktoré majú definovanú váhu. Používame ich napr. pri určení toku v sieti, alebo jednoducho ak chceme hrane pripísať určitú hodnotu, ktorá bude označovať jej dôležitosť.



Obr. 1: Rôzne typy grafov. A - Multigraf, B - Prostý graf, C - Obyčajný graf



Obr. 2: Ukážka zakreslenia ohodnotenej orientovanej hrany.

- *Neohodnotené* – hrana nemá definovanú váhu.

Neohodnotené neorientované hrany sú použité na obr. 1. Na obr. 2 je ukážka zakreslenia ohodnotenej orientovanej hrany.

Analogicky nazývame grafy, podľa toho aké typy hrán obsahujú. V práci sa budeme pri experimentoch zaoberať iba *neohodnotenými neorientovanými prostými grafmi* nazývanými tiež ako *binárne grafy*.

Formálne môžeme graf definovať nasledovne :

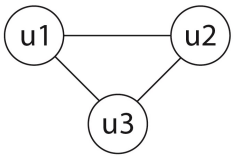
Definícia 2.1 *Jednoduchý graf G je usporiadaná dvojica (V,E), kde V (z angl. vertex) je neprázdna množina vrcholov a E (z angl. edge) je nejaká množina dvojprvkových podmnožín množiny V. Prvkom E hovoríme hrany. [44].*

Množinu vrcholov grafu G označujeme $V(G)$ a množinu hrán $E(G)$, vystačíme si však aj s označením V a E. Celkovo budeme graf označovať zápisom $G(V, E)$. Počet hrán a vrcholov budeme vyjadrovať ako $e(G)$ a $v(G)$, resp. $|E|$ a $|V|$

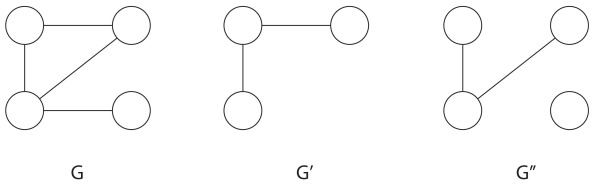
Pre graf, ktorého všetky vrcholy sú susedné (má pri danom počte vrcholov maximálny počet hrán), sa používa označenie *úplný graf*. Graf, ktorého vrcholy majú rovnaký stupeň nazývame *pravidelný*, alebo *regulárny*. Na obr. 3 je teda úplný regulárny graf.

Všetky grafy môžu byť rozdelené do dvoch kategórií a to na *riedke* a *husté* grafy. Riedke grafy obsahujú malý počet hrán, rádovo $|E| << |V|^2$. Husté grafy naopak obsahujú približne rovnaký počet hrán ako $|V|^2$.

Hrany sú definované dvojicami koncových vrcholov, preto konkrétnu hranu budeme zapisovať ako dvojprvkovú množinu jej vrcholov. Napr. pre graf na obr. 3 vyššie, by sme hrany zapísali nasledovne $E = \{\{u1, u3\}, \{u1, u2\}, \{u2, u3\}\}$. Pri praktických experimentoch budeme pracovať so vstupnými súbormi, v ktorých sú grafy reprezentované práve pomocou dvojprvkových množín vrcholov.



Obr. 3: Úplný regulárny graf



Obr. 4: G′ a G″ - podgrafy grafu G

Ďalej pre stručnosť uvedieme len definície a vety, ktoré budú užitočné pre túto prácu. Budú sa teda týkať *neohodnotených neorientovaných grafov*, alebo grafov vo všeobecnosti. Čerpali sme z [45].

2.1.2 Podgraf a klika grafu

Definícia 2.2 *Nech je daný graf G(V,E). Potom graf G′(V′, E′) taký, že :*

- *V′ je podmnožina V*
- *E′ je podmnožina E*

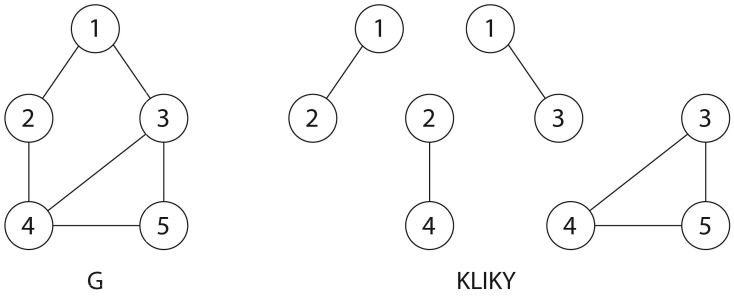
nazveme ***podgrafom*** *grafu G.*

Definícia 2.3 ***Klikou*** *grafu G(V,E) nazveme každý jeho úplný podgraf.*[1]

Definícia 2.4 ***Maximálna klika*** *grafu G(V,E) je klika, ktorá nemôže byť rozšírená pridaním ďalšieho vrcholu (nie je obsiahnutá v inej klike).*

Definícia 2.5 ***K-klikou*** *grafu G(V,E) nazveme každý jeho úplný podgraf, ktorý ma práve* k *vrcholov (nemusí byť maximálny).*

[1] V literatúre sa často pojmom „klika“ označuje maximálna klika.



Obr. 5: Graf G a jeho 4 kliky.

2.1.3 Stupeň vrcholu

Definícia 2.6 *Stupeň vrcholu je počet hrán, ktoré sú s týmto vrcholom spojené – majú tento vrchol ako koncový.*

Pre skutočnosť, že hrana má daný vrchol ako koncový, používame termín, že hrana s týmto vrcholom *inciduje*. Pre označenie stupňa vrcholu *u* používame zápis *d(u)*. Písmeno *d* je z anglického slova degree – stupeň.

2.1.4 Sled, ťah, cesta, konektivita grafu

Definícia 2.7 *Nech je daný graf G(V,E) a dva jeho vrcholy* u *a* v. ***Sledom*** *medzi vrcholmi* u *a* v *nazveme postupnosť vrcholov a hrán* $u, e_{i1}, u_{i1}, e_{i2}, u_{i2}, ..., u_{ik-1}, e_{ik}, v$.

Sled je teda na seba nadväzujúca postupnosť hrán, kde vždy dve za sebou nasledujúce hrany v slede majú spoločný koncový vrchol, ktorý je v slede uvedený medzi nimi.

Definícia 2.8 ***Ťah*** *medzi vrcholmi* u *a* v *je sled medzi týmito dvoma vrcholmi, v ktorom sa žiadna hrana nevyskytuje viackrát.*

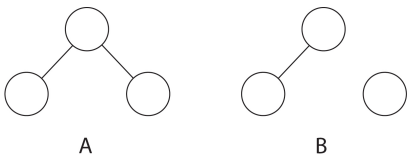
Definícia 2.9 ***Cesta*** *medzi vrcholmi* u *a* v *je ťah (def. 2.8) medzi týmito dvoma vrcholmi, v ktorom sa žiaden jeho vnútorný uzol nevyskytuje viackrát.*

Vzdialenosť medzi dvoma vrcholmi *u* a *v* grafu meriame ako dĺžku najkratšej cesty (def. 2.9) medzi nimi. Budeme ju značiť *dist(u,v)*. So vzdialenosťami je spojený aj pojem *excentricita*.

Definícia 2.10 *Majme graf* G *a vrchol* $v \in V(G)$. ***Excentricita*** *vrcholu* v *je najväčšia zo všetkých vzdialeností* v *od ostatných vrcholov v* G. *Excentricitu značíme* ecc(v), *prípadne* $ecc_G(v)$. *Symbolicky môžeme excentricitu vrcholu v popísať ako*

$$ecc(v) = max_{u \in V(G)}\{\text{dist}(u, v)\}$$

Definícia 2.11 *Priemer grafu G, označovaný* ***diam(G)****, je rovný najväčšej excentricite (def. 2.10).*



Obr. 6: Súvislý (A) a nesúvislý (B) graf

Definícia 2.12 *Graf, medzi ktorého každými dvoma vrcholmi existuje cesta, nazývame* ***súvislým grafom****.*

Definícia 2.13 ***Komponentom*** *grafu G nazveme každý jeho* ***maximálny súvislý podgraf*** *(def. 2.12).*

Pritom súvislý podgraf grafu G považujeme za maximálny, ak sa už nedá zväčšiť pridaním ďalších hrán, či uzlov z grafu G tak, aby podgraf ostal stále súvislý. Teda nie je vlastným podgrafom iného súvislého podgrafu.

V práci budeme hovoriť o súvislosti ako o konektivite grafu. Konektivita grafu sa dá overiť napríklad prehľadávaním grafu do šírky.

2.1.5 Prehľadávanie grafu do šírky - BFS

Tento algoritmus nám umožňuje vyhľadať všetky vrcholy, ktoré sú dosiahnuteľné z daného počiatočného vrcholu. BFS prechádza súvislý komponent grafu a vytvára jeho kostru. V jednoduchosti sa dá povedať, že základnou myšlienkou BFS je vyslať akúsi „vlnu“ od počiatočného vrcholu. Táto vlna najskôr prejde všetkými vrcholmi, ktoré sú od počiatočného vrcholu vzdialené o 1 hranu (sú susedmi počiatočného vrcholu). Potom prejde všetkými vrcholmi, ktoré sú od počiatočného vzdialené o 2 hrany, atď. Dá sa teda povedať, že prechádzame všetkých susedov počiatočného vrcholu, potom susedov ich susedov atď.

Ako dátovú štruktúru využitú na správu týchto „vĺn“ používame *frontu*, označme ju Q. Vrchol v sa nachádza vo fronte Q len vtedy ak sa „vlna“ dostala do v, ale ešte z neho nevyšla (v ešte nebol spracovaný). Keďže Q pracuje na princípe *FIFO*, tak sú vrcholy spracúvané v takom poradí, v akom boli vložené do fronty.

Pseudokód pre nájdenie komponentu súvislosti pomocou BFS :

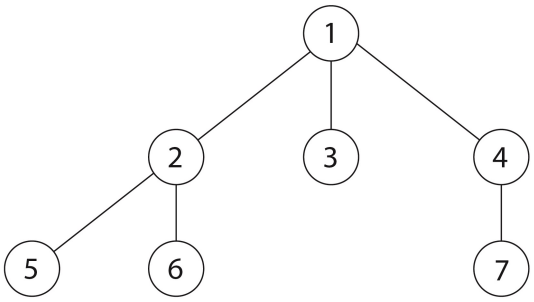
Vrcholy grafu na obr. 7, by sa pri spustení BFS z vrcholu 1 spracúvali v poradí, v akom sú očíslované na obrázku.

2.1.6 Reprezentácia grafov v PC

Na to, aby sme mohli graf algoritmicky analyzovať, ho musíme v počítači vhodne reprezentovať. Existuje niekoľko základných štruktúr pre reprezentáciu grafu v PC. Uvedieme len verzie pre neohodnotené neorientované grafy.

Pseudokód 1 BFS(*G,s*)

```
Require: s - počiatočný vrchol, G - graf ktorý prechádzame
  Vytvor frontu Q
  Vytvor prázdny komponent súvislosti - List C
  Q ← s
  while Q ≠ empty do
    aktuálnyVrchol = Q.poll();
    for all v : aktuálnyVrchol.getSusednéVrcholy() do
      if v ∉ C and v ∉ Q then
        Q ← v
      end if
    end for
    C ← aktuálnyVrchol
  end while
```



Obr. 7: Príklad BFS.

2.1.6.1 Incidenčná matica (angl. incidence matrix) Majme daný graf *G*. *Incidenčná matica B(G)* je obdĺžniková matica s *v(G)* riadkami a *e(G)* stĺpcami. Vrcholy grafu *G* označíme $v_1, v_2, ..., v_n$ a hrany $e_1, e_2, ..., e_m$. Každému vrcholu grafu *G* odpovedá jeden riadok matice *B* a každej hrane grafu *G* odpovedá jeden stĺpec matice *B*. Prvok b_{ij} matice *B* nadobúda hodnotu 1 práve vtedy, keď je vrchol v_i incidentný s hranou e_j. V opačnom prípade je $b_{ij} = 0$. Súčet čísel v každom stĺpci tejto matice dáva 2 a súčet čísel v *i*-tom riadku dáva stupeň (def. 2.6) vrcholu v_i. Incidenčná matica je preto veľmi rozsiahla a riedka. Obsahuje len *2m* jednotiek z celkového počtu mn prvkov [44].

Príklad incidenčnej matice pre obr. 7, str. 13 je :

$$B(G) = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \end{array} \begin{array}{c} \begin{array}{cccccc} v_1v_2 & v_1v_3 & v_1v_4 & v_2v_5 & v_2v_6 & v_4v_7 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

Tento spôsob reprezentácie grafu je kvôli pamäťovým nárokom najmenej používaný.

2.1.6.2 Matica susednosti (angl. adjacency matrix) O niečo úspornejšie využitie pamäte pre husté grafy získame uložením grafu pomocou *matice susednosti*. Značenie vrcholov ponecháme rovnaké ako v predchádzajúcom príklade. Matica susednosti *A(G)* je štvorcová matica rádu *n*, v ktorej je prvok $a_{ij} = 1$ práve vtedy ak medzi vrcholmi v_i a v_j existuje hrana. Takáto matica je pre jednoduché grafy symetrická a platí, že súčet čísel v *i*-tom riadku (*i*-tom stĺpci) je rovný stupňu vrcholu v_i [44].

Matica susednosti grafu z obr. 7, str. 13 je :

$$A(G) = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \end{array} \begin{array}{c} \begin{array}{ccccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

Veľmi veľkou výhodou tejto reprezentácie je, že odstránenie, či pridanie hrany môže byť vykonané v konštantnom čase *O(1)*. Rovnako rýchlo môžeme zistiť, či medzi danými dvoma vrcholmi existuje hrana.

Medzi nevýhody patrí to, že matica susednosti (aj keď môže byť úspornejšia ako incidenčná matica) zaberá veľkú časť pamäte pokiaľ sa chystáme spracovávať obrovské grafy. Ďalšou nevýhodou je to, že v mnohých algoritmoch potrebujeme získať zoznam susedov konkrétneho vrcholu v_i. Aby sme takúto informáciu z matice susednosti dostali, musíme prejsť všetky stĺpce *i*-teho riadku, čo znamená náročnosť $O(|V|)$. Pritom, za pomoci *zoznamu susedov* (sek. 2.1.6.3) môžeme túto náročnosť zredukovať na *O(1)*. Poslednou nevýhodou matice susednosti je, že dodatočné odstránenie/pridanie vrcholu je veľmi náročné.

2.1.6.3 Zoznam susedov (angl. adjacency list) Graf môžeme reprezentovať aj pomocou zoznamov susedných vrcholov. Pre každý vrchol grafu v_i, môžeme vytvoriť pole a_i jeho susedných vrcholov. Každé pole a_i má potom počet položiek na základe stupňa vrcholu v_i.

Graf z obr.7, str. 13 by bol potom reprezentovaný nasledovne :
$a_1 = [v_2, v_3, v_4]$
$a_2 = [v_1, v_5, v_6]$
$a_3 = [v_1]$
$a_4 = [v_1, v_7]$
$a_5 = [v_2]$
$a_6 = [v_2]$
$a_7 = [v_4]$
Ak má graf *h* hrán, potom budú zoznamy obsahovať celkom *2h* položiek, pretože každá hrana je uložená 2 krát [44].

Výhodou tejto štruktúry je to, že sa dá počas behu algoritmu ľahko modifikovať. Nevýhodou je naopak väčšia časová zložitosť pri zisťovaní existencie konkrétnej hrany, v čom naopak vyniká matica susednosti. Vyhľadávanie sa však dá urýchliť ak prvky v zoznamoch usporiadame podľa pevne zvoleného kľúča a budeme používať *binárne vyhľadávanie*.

2.2 Vlastnosti komplexných sietí

Ešte predtým, ako prejdeme k rozdeleniu komplexných sietí v stručnosti uvedieme niektoré z ich základných vlastností. Týmito vlastnosťami sa líšia od jednoduchých grafov, ako sú regulárne grafy (def. 2.1, str. 9) a pod.

Reálne komplexné siete sú typické tým, že majú často veľmi malý *priemer* – jedná sa o tzv. *fenomén malého sveta*. Ďalej je pre ne charakteristické to, že ich *distribúcia stupňov* sa riadi *mocninovým rozdelením*. Existuje v nich mnoho vrcholov, ktoré majú malý stupeň ale nájde sa aj pár takých, ktoré majú naopak veľmi veľký stupeň. V reálnych sieťach pozorujeme tiež *homofýliu (assortative mixing)* – napr. sebe podobný ľudia (vekom, rasou, ...), alebo články zamerané na rovnaké téma, majú tendenciu vytvárať skupiny. Mnoho reálnych sietí má vysoký *zhlukovací koeficient*.

Zhlukovací koeficient C [1] meria hustotu trojuholníkov (trojica vrcholov, kde je každý vrchol spojený so zvyšnými dvoma) v grafe. Jedná sa o to, že v reálnych sieťach je veľmi pravdepodobné, že ak je spojený vrchol *a* s vrcholom *b* a *b* je zároveň spojený s vrcholom *c*, tak bude spojený aj *c* s *a*. Zhlukovací koeficient *C* grafu *G* môžeme vypočítať ako :

$$C = \frac{\textit{3*počet trojuholníkov v G}}{\textit{počet trojíc v G}}$$

Vlastnosti sietí zachytávajú rôzne miery a metriky (často pochádzajú z oblasti sociológie) ako napr. :

- *Centralita* – skúma, ktoré vrcholy sú v sieti najdôležitejšie. Existujú rôzne prístupy - *PageRank, Betweeness, Closeness, Degree, ...*
- *Štrukturálna vyváženosť* – ak sú hrany v sieti označené ako pozitívne a negatívne, tak je sieť štrukturálne vyvážená pokiaľ obsahuje nepárny počet negatívnych hrán.

- *Podobnosť* – označuje štrukturálnu podobnosť medzi vrcholmi. Meria sa napr. na základe počtu spoločných susedov.

Skúmaných metrík a vlastností existuje ešte omnoho viac, uviesť ich všetky je však mimo rozsah tejto práce. Niektoré spomenieme ešte v priebehu ďalšieho textu. Viac sa dá dozvedieť v [1].

2.3 Typy komplexných sietí

Vo všeobecnosti môžeme komplexné siete rozdeliť do štyroch kategórií, ktoré použil v [1] Newman :

- sociálne
- informačné
- technologické
- biologické

Poďme sa bližšie pozrieť na každý z týchto typov.

2.3.1 Sociálne siete

Sociálna sieť je množina ľudí, alebo skupín ľudí, medzi ktorými existuje určitý druh prepojenia, resp. interakcie. Môže sa jednať napr. o priateľstvo, pracovný vzťah, príbuzenský vzťah alebo môže ísť len o to, že jedna osoba poslala druhej mailovú správu.

Za jedny z najznámejších experimentov v oblasti sociálnych sietí (nie tých na webe) sa považujú Milgramove experimenty *malého sveta* [29] [30]. Princíp experimentov spočíval v tom, že Milgram napísal veľké množstvo dopisov, ktoré rozdal účastníkom experimentu. Každý z nich mal dopis poslať svojim známym tak, aby sa dostal k Milgramovmu priateľovi v Bostone. Veľké množstvo dopisov sa počas experimentu stratilo, ale asi štvrtine sa podarilo dosiahnuť cieľ. Zaujímavé bolo, že každý z nich prešiel v priemere, len rukami šiestich ľudí. Tento fakt dal za vznik populárnej teórii s názvom *šesť stupňov odlúčenia*, hoci sám Milgram tento názov nikdy nepoužil.

Typicky nás pri analýze sociálnych sietí zaujímajú odpovede na problém *centrality* (ktorý jedinci sú najlepšie prepojení s ostatnými, alebo majú najväčší vplyv) a *konektivity* (či a ako sú jedinci vzájomne v sieti spojení). Tradičné štúdie sociálnych sietí však často trpia problémom malej testovacej vzorky, nepresnosti a subjektivity. Až na pár štúdií, ako boli tie Milgramove, sa dáta získavajú pomocou dotazníkov, čo je veľmi prácne a nepresné. To, ako jeden človek chápe priateľstvo, sa vôbec nemusí zhodovať s predstavou priateľstva iného človeka. Kvôli týmto problémom sa vedci zamerali na štúdium iných sociálnych sietí, ako sú napr. siete spoluprác. Príkladom takejto siete je napr. sieť spolupráce filmových hercov. Vrcholy siete sú predstavované hercami a hrana medzi nimi znamená, že spolu hrali v nejakom filme. Do sociálnych sietí radíme aj siete komunikačné, kde medzi jedincami vzniká hrana, ak si napr. poslali dopis, balík, email, alebo spolu

uskutočnili telefónny hovor. Možností je mnoho, záleží od toho, aký druh komunikácie chceme skúmať.

V neposlednom rade sem patria aj webové sociálne siete ako je napr. Facebook, Twitter, či LinkedIn. Práve tieto sa stali v poslednej dobe, vďaka ľahkému získaniu presných dát, cieľom mnohých analýz.

2.3.2 Informačné siete

Tiež nazývané ako *vedomostné siete*. Typickým príkladom tejto kategórie je sieť citácií medzi akademickými publikáciami, ktorú budeme mimochodom analyzovať v experimentálnej časti práce (sek. 6.4, str. 65). Hrana medzi dvoma publikáciami, ktoré predstavujú vrcholy, vzniká ak sa v jednej práci nachádza citácia na tú druhú. Podľa toho, ako chceme sieť skúmať, môžeme takúto hranu uvádzať ako orientovanú, alebo ako neorientovanú. Štruktúra citačnej siete odráža štruktúru informácií, ktoré sú uchované v jej vrcholoch, preto o nej hovoríme ako o informačnej sieti [1].

Ďalším veľmi dôležitým príkladom informačnej siete je WWW, čo je, sieť webových stránok prepojených hyperlinkami. Veľmi často je nesprávne označovaná ako Internet, čo je sieť fyzicky prepojených počítačov. Medzi prvé štúdie WWW patria napr. [32] [33].

2.3.3 Technologické siete

Jedná sa o človekom vytvorené siete, typicky slúžiace na rozvoz nejakej komodity alebo zdroja, ako je napr. elektrická energia, alebo informácie. Príkladmi takýchto sietí sú: elektrická rozvodná sieť, vlakové, letecké, cestné a iné dopravné komunikácie, telefónne siete (myslíme fyzickú telefónnu sieť, nie sieť hovorov) a iné. V neposlednom rade sem patrí aj spomínaný Internet.

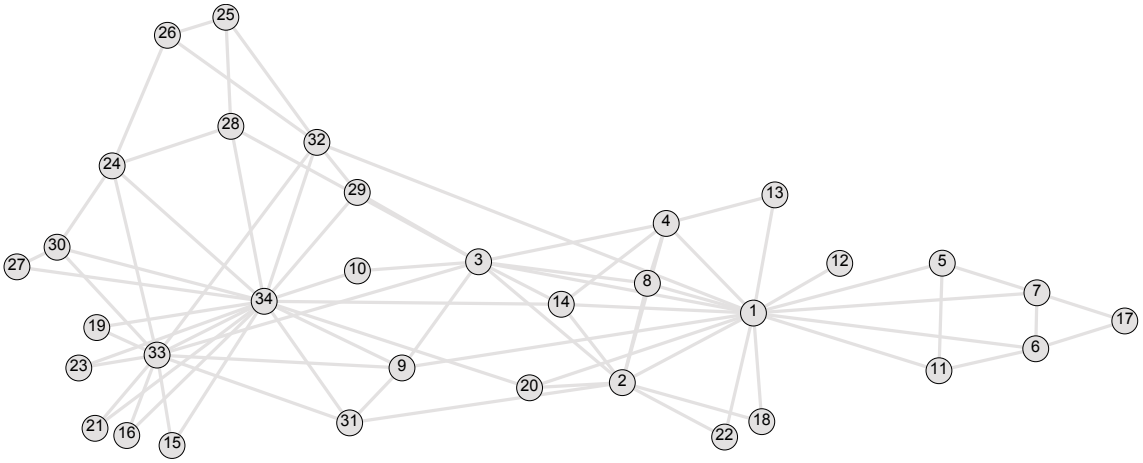
Štúdie týchto sietí nájdeme napr. v [34] [35] [36] [37].

2.3.4 Biologické siete

Sú posledným typom sietí, z toho ako ich popisuje Newman [1]. Množstvo biologických systémov môže byť užitočné reprezentovať ako sieť. Klasickým príkladom biologickej siete je sieť metabolických ciest. Je to reprezentácia metabolických substrátov a produktov, s orientovanými hranami, ktoré ich spájajú, pokiaľ existuje metabolická reakcia, ktorá prebieha na danom substráte a vytvára daný produkt. Štúdie štatistických vlastností metabolických sietí boli vykonané napr. v [38] alebo v [39]. Inými príkladmi biologických sietí, sú napr. sieť interakcie medzi proteínmi, sieť potravinového reťazca zachytávajúca vzťah lovec-korisť medzi rôznymi ekosystémami, alebo neurologické siete a iné.

2.4 Analýza sietí

Už v úvode sme naznačili, prečo je analýza dôležitá. Pre motiváciu uvedieme ešte jednu, možno trochu prehnane povedané, katastrofickú situáciu, ktorá sa týka topológie komplexnej siete.



Obr. 8: Sieť členov Zacharyho karate klubu, ktorý sa neskôr rozpadol na 2 časti.

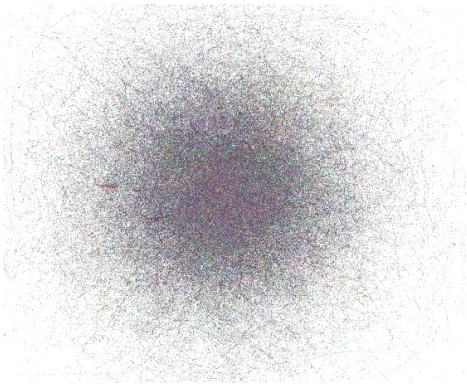
10. Augusta 1996 spôsobila chyba na dvoch častiach elektrickej rozvodovej siete v Oregone kaskádu zlyhaní, ktoré viedli k totálnemu výpadku dodávky elektrickej energie v 11 štátoch USA a 2 Kanadských provincií. 7 miliónov obyvateľov vtedy ostalo bez prúdu na celých 16 hodín [2].

Práve skúmaním topologických vlastností sietí môžeme podobným situáciám predísť. Je nutné podotknúť, že prístup k analýze sa v posledných rokoch rapídne zmenil. V počiatkoch tohto oboru bolo nepredstaviteľné skúmať grafy, ktoré mali počet vrcholov v rádoch tisícov. Väčšinou sa skúmali len, na dnešnú dobu, malé grafy v rádoch desiatok až stoviek vrcholov, pričom sa analýza sústredila na vlastnosti jednotlivých vrcholov [1].

Pri tak malých sieťach sme si mohli klásť otázky typu : „Ktorý vrchol je najdôležitejší z pohľadu zachovania konektivity siete pri jeho prípadnom odstránení ?“ . Avšak pri sieťach, ktoré majú obrovský počet vrcholov je takáto otázka zväčša irelevantná, pretože sa nestáva, že odobranie jedného vrcholu má globálny dopad na celú sieť. Momentálne sa skôr pýtame v štýle : „Koľko percent vrcholov musíme odobrať aby sme ovplyvnili konektivitu siete v nejakom smere ?“. No nielenže sa z časti zmenilo to, čo nás pri analýze sietí zaujíma, ale čo je hlavné, výrazne sa zmenil aj samotný spôsob akým siete analyzujeme. Zo začiatku nebol problém menšie grafy vizualizovať, teda nakresliť obrázok siete s vrcholmi a hranami a analýzu vykonávať jednoducho pohľadom. Na obr. 8 je ukážka zobrazenia sociálnej siete členov Zacharycho karate klubu. Táto sieť bude použitá aj pri experimentoch v závere práce (sek. 6.2, str. 61).

Je vidieť, že malá sieť (táto má 34 vrcholov a 78 hrán) sa dá analyzovať aj voľným okom, i keď je to prácne.

Sieť s miliónmi vrcholov by sme sa však pokúšali nakresliť a analyzovať pohľadom márne. Samozrejme, že v dnešnej dobe existujú algoritmy, ktoré zvládnu vykresliť aj obrovské siete, ako napr. *Yi Fan Hu*, *OpenORD* a iné. No ak nie je toto vykreslenie doplnené napr. o vyfarbenie vrcholov podľa toho do akej komunity patria, alebo zobrazenie veľkosti vrcholu podľa jeho dôležitosti a pod., tak nám kvôli svojmu rozsahu neposkytne



Obr. 9: Sieť vedeckých článkov.

žiadne informácie. Na obr. 9 je ukázaná vizualizácia citačnej siete vedeckých publikácií. Je vidieť, že aj keď máme k dispozícii vyfarbené vrcholy podľa ich členstva v komunite, analýza voľným okom je pri takto veľkých sieťach (52 615 vrcholov a 65 000 hrán) prakticky nemožná.

Komplexné siete už neanalyzujeme pohľadom, ale algoritmicky, skúmaním rôznych štatistických vlastností, ako je napr. dĺžka ciest medzi vrcholmi, alebo rozdelenie stupňa uzlov. Veľmi dôležitou a netriviálnou súčasťou analýzy sietí je aj **detekcia komunít**.

3 Komunity v komplexných sieťach a ich detekcia

Ako sme už spomenuli, reálne komplexné siete majú, oproti jednoduchým grafom, mnoho zaujímavých vlastností (sek. 2.2, str. 15). Asi najzásadnejšou z nich je to, že obsahujú komunitné štruktúry, zjednodušene *komunity (klastre, skupiny, moduly)*. Problémom je, že neexistuje presná definícia komunity, ktorá by bola univerzálne akceptovateľná. Definícia komunity totiž záleží od povahy siete, ktorú skúmame, alebo od povahy samotnej aplikácie, resp. použitého algoritmu. Intuitívne však môžeme komunitu chápať ako *zoskupenie vrcholov, ktoré sú medzi sebou prepojené viac, ako so zvyškom grafu*. Dôležitou vlastnosťou komunity je to, že musí byť súvislá (def. 2.12, str. 15). Toto sú základné myšlienky, od ktorých sa odvíjajú iné, špecifickejšie definície. Príklad komunít v grafe je na obr. 10.

Na základe vyššie uvedeného, je základný formálny popis komunity Cnasledovný. Nech C je podgrafom grafu G s $|C| = n_c$ a $|G| = n$ vrcholmi. Definujme *interný* a *externý* stupeň vrcholu $v \in C, k_v^{int}$ a k_v^{ext}, ako počet hrán, ktoré spájajú v s ostatnými vrcholmi komunity C alebo so zvyškom grafu G. Ak je $k_v^{ext} = 0$, tak má vrchol v susedov len v C, čo značí, že C je pre v dobre určená komunita. Ak je $k_v^{int} = 0$, tak to znamená, že v nie je spojený s C a mal by byť priradený do inej komunity. *Interný stupeň* k_{int}^C komunity C je súčet interných stupňov jej vrcholov. Podobne *externý stupeň* k_{ext}^C komunity C je súčet externých stupňov jej vrcholov. *Celkový stupeň* k^C je súčet stupňov vrcholov $\in C$. Potom $k^C = k_{int}^C + k_{ext}^C$ [2].

Definujeme *internú hustotu* $dens_{int}(C)$ komunity C ako pomer medzi počtom hrán interných pre C a maximálnym možným počtom hrán interných pre C [2].

$$dens_{int}(C) = \frac{\#\ hrany\ interné\ pre\ C}{n_c(n_c - 1)/2} \tag{1}$$

Hrana je interná pre C ak oba jej koncové vrcholy patria C.

Podobne definujeme *externú hustotu* $dens_{ext}(C)$ ako pomer medzi počtom hrán, spájajúcich C so zvyškom grafu a celkovým možným počtom takýchto hrán (ext. hrany).

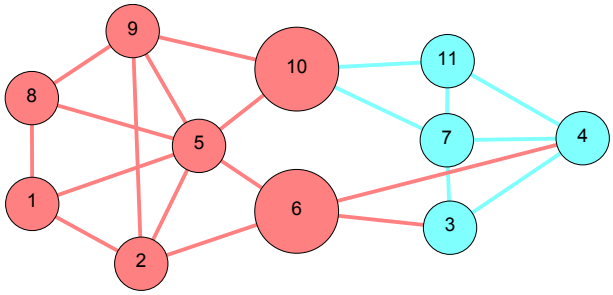
$$dens_{ext}(C) = \frac{\#\ ext.\ hrany\ C}{n_c(n - n_c)} \tag{2}$$

Na to, aby bola C komunita grafu $G(V, E)$, intuitívne očakávame, že $dens_{int}(C)$ bude väčšia ako priemerná hranová hustota grafu $dens(G)$.

$$dens(G) = \frac{|E|}{n(n-1)/2} \tag{3}$$

Na druhú stranu, by mala byť $dens_{ext}(C)$ o mnoho menšia ako $dens(G)$. Hľadanie vhodného kompromisu medzi veľkým $dens_{int}(C)$ a malým $dens_{ext}(C)$ je implicitným, alebo explicitným cieľom väčšiny algoritmov na vyhľadávanie komunít.

Môžeme ešte uviesť definíciu *silnej* a *slabej* komunity, tak ako ich opísal Raddicchi *et al.* v [13].



Obr. 10: Príklad dvoch, farebne odlíšených, komunít v grafe.

Definícia 3.1 *Podgraf C predstavuje silnú komunitu ak platí že :*

$$k_i^{in}(C) > k_i^{out}(C), \forall i \in C. \tag{4}$$

V silnej komunite má každý vrchol viac hrán vo vnútri komunity, ako so zvyškom grafu.

Definícia 3.2 *Podgraf C predstavuje slabú komunitu ak platí že :*

$$\sum_{i \in C} k_i^{in}(C) > \sum_{i \in C} k_i^{out}(C). \tag{5}$$

V slabej komunite je súčet všetkých vnútorných stupňov väčší (vnútorné hrany pre každý vrchol), ako súčet stupňov so zvyškom grafu.

Metódy na detekciu komunít používajú mnoho rôznych prístupov. Na základe toho, o akú definíciu komunity sa opierajú, môžu odhaľovať klastre, ktoré:

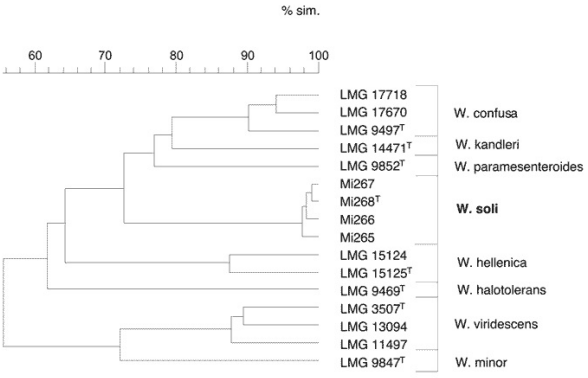
- maximalizujú nejakú *globálnu funkciu* kvality,
- obsahujú *špecifické sub-štruktúry*,
- alebo najviac vyhovujú určitým *lokálnym kritériám* [2].

Tieto prístupy sa potom delia na základe toho, či umožňujú, aby sa jednotlivé komunity *prekrývali*, alebo nie (sek. 3.1).

Ďalej sa v tejto kapitole budeme venovať popisu vyššie spomínaných prístupov a niektorých vybraných metód, ktoré ich reprezentujú. Globálnym metódam (sek. 3.2) sa budeme venovať len v stručnosti, pretože nie sú náplňou tejto práce. Avšak je vhodné, aby si o nich čitateľ spravil aspoň základnú predstavu a pochopil tak rozdiely medzi jednotlivými prístupmi.

3.1 Prekrývajúce sa vs. neprekrývajúce sa komunity

Odpoveď na otázku toho, či sa komunity majú prekrývať, alebo nie, je asi najpodstatnejšia vec v oblasti detekcie komunít. V minulosti boli vyvíjané metódy, ktoré s prekrytím



Obr. 11: Dendogram odvodený z analýzy proteínových vzorov *Weissella soli* [40]

.

nepracovali, tzn. že každý vrchol patril vždy do jednej komunity. S nárastom on-line komunikácie a explóziou webových sociálnych sietí, sa však vedcom sprístupnili dáta, v ktorých je prekrývanie komunít prirodzené.

Príklad prekrývajúcich sa komunít je, už na spomenutom, obr. 10. Vrcholy č. 6 a č. 10 patria do oboch farebných komunít, čo je znázornené tým, že majú väčšiu veľkosť. Metóda, ktorá detekuje neprekrývajúce komunity, by ich zaradila tak, ako je to znázornené farebne (s odmyslením inej veľkosti).

Poďme sa na jednotlivé prístupy pozrieť bližšie.

3.1.1 Detekcia neprekrývajúcich sa komunít

Veľmi veľké množstvo metód (hlavne globálne, sek. 3.2) pristupuje k problému detekcie komunít ako k problému hierarchického delenia grafu. Podľa tohto prístupu sa predpokladá, že štruktúra siete je hierarchická. Znamená to, že jedinci postupne vytvárajú skupiny, ktoré sa stávajú podskupinami ďalších skupín, až sa vytvorí jedna veľká skupina, ktorá obsahuje celý graf (alebo opačne). Takéto hierarchické spájanie (alebo delenie) môžeme znázorniť pomocou dendogramu. *Dendogram* je graf, ktorý znázorňuje stromovú štruktúru delenia, viď obr. 11.

Dôležité je si uvedomiť, že pri takomto postupe, patrí vrchol vždy, len do jednej komunity. Pri niektorých typoch sietí však presne toto požadujeme. Medzi ne patria napr. organizačné siete, alebo taxonómie. Konkrétny príklad je napr. keď analyzujeme biologické siete interakcie proteínov a chceme vygenerovať ich taxonómiu (obr. 11). Inokedy naopak chceme detekovať prekrývajúce sa komunity.

3.1.2 Detekcia prekrývajúcich sa komunít

Hlavne pri sociálnych sieťach by sme kvôli zaraďovaniu vrcholu len do jednej komunity prišli o veľa podstatných informácií. Jednoduchým príkladom je situácia, kedy by sme chceli vyhľadať komunity v grafe, ktorý reprezentuje ľudí navštevujúcich záujmové

krúžky. Vrcholy grafu sú ľudia a hrana medzi vrcholmi znamená, že navštevujú spoločný krúžok. Ľudia v jednom krúžku tvoria komunitu. Je úplne bežné, že človek môže patriť do viacerých záujmových krúžkov, tj. do viacerých komunít. Metódy detekujúce neprekrývajúce sa komunity, by tento fakt nezohľadnili. Preto je na podobné typy sietí, oveľa lepšie použiť detekciu prekrývajúcich sa komunít.

Metódy, ktoré zohľadňujú tento typ komunít, dovoľujú patriť jednému vrcholu do viacerých skupín. Často krát to má, u metód s heuristickým prístupom (sek. 3.3, str. 24), za následok to, že vo výsledku dostaneme veľké množstvo, veľmi sa prekrývajúcich komunít. Tento problém potom musíme odstraňovať vhodným post/pre-processingom daných dát.

3.2 Globálne metódy

Tieto metódy vyhľadávajú komunity na základe nejakej globálne definovanej funkcie, resp. globálneho kritéria. Z toho vyplýva, že potrebujú mať dostupné informácie o celom grafe (o jeho štruktúre). V prípade veľkých sietí, ako je napr. WWW, sa jedná o nesplniteľnú požiadavku. Ak by sme pomocou nich analyzovali len časť siete, znamenalo by to, že globálna funkcia by bola definovaná pre túto časť. Mohli by sme teda dostať výsledky, ktoré neodrážajú skutočnosť. Po rozšírení analyzovanej časti grafu, by sa totiž zmenila hodnota globálnej funkcie pre jednotlivé vrcholy a tie by sa mohli zaradiť do iných komunít ako pôvodne (v prípade CPM, sek. 4.1, str. 34), by tento problém nenastal, preto má z časti lokálnu povahu).

Tradiční zástupcovia týchto metód (viď napr. [21, 22]) fungujú na princípe delenia grafu, alebo hierarchického zhlukovania (obr. 11) s využitím nejakej miery/metriky (podobnosť, closeness centralita, ...). Niektoré z týchto metód potrebujú vopred poznať počet komunít, ktoré majú byť nájdené. Ďalším, už spomenutým, obmedzením pri delení a hierarchickom zhlukovaní je, že odhaľujú len neprekrývajúce sa komunity. Taktiež je väčšina z nich výpočtovo náročná (zvyčajne *NP-úplné*) [2].

Pôvodne sa určovala hierarchická štruktúra tak, že sa opakovane identifikovali hrany, ktoré nepatrili do rovnakého hustého podgrafu [21] [22]. Majme skupinu, ktorá obsahuje všetky vrcholy (celý graf) a pre každú hranu v tejto skupine vypočítajme centralitu na základe niektorej z definícií (closeness, betweenness, ...). Hrany s vysokou hodnotou centrality budú tie, ktoré spájajú jednotlivé klastre. Takéto hrany sa opakovane odstraňujú, čím vznikajú oddelené podgrafy, až kým sa nedostaneme do bodu, že každý vrchol je v skupine sám (viď obr. 11). Ako výsledok je vygenerovaná hierarchia „rozdelení“, ktorá ukazuje vzťah medzi menšími a väčšími skupinami. Problémom je, že nevieme posúdiť, v ktorom bode rozdeľovania sme sa dopracovali k „najlepším“ komunitám.

Z dôvodu automatizovaného určenia najlepšieho „rozdelenia“ bol zavedený pojem *modularity* [23]. Modularita Q je vyjadrená ako

$$Q = \frac{1}{2m} \sum_{i,j \in V} \left[A_{i,j} - \frac{k_i k_j}{2m} \right] \delta(c_i c_j) \tag{6}$$

kde m je počet hrán siete, $A_{i,j}$ je váha hrany, ktorá spája vrcholy i a j (resp. 0 / 1 v príp. neohodnotených grafov), k_i a k_j sú stupne vrcholov i a j a $\delta(c_i c_j)$ je funkcia, ktorá vracia

1 ak sú vrcholy i a j zaradené do rovnakej komunity (c_i, c_j sú ich komunity), inak vracia 0. Táto miera v podstate vyjadruje aký je rozdiel medzi reálnym počtom hrán v komunite a očakávaným počtom hrán v komunite, na základe stupňa komunity. Môžeme porovnať hodnoty modularity všetkých úrovní hierarchie a identifikovať najlepšiu skupinu komunít, vzhľadom na *null model* [2]. *Null model* predstavuje náhodná sieť, s pevne daným stupňom vrcholov. Je vygenerovaná postupným spájaním náhodných vrcholov, s tým, že sa berie ohľad na preddefinovaný stupeň vrcholu.

Po zavedení modularity vzniklo mnoho metód, ktoré s ňou pracujú. Asi najznámejšou je greedy aglomeračná metóda, publikovaná v [24]. Tento algoritmus pracuje tak, že vrcholy na začiatku priradí do jedinečných komunít, ktoré postupne zlučuje na základe najväčšieho navýšenia modularity. Ďalšie metódy, ktoré vyhľadávajú komunity, na základe modularity sú napr. [25], [26] a iné.

Medzi globálne metódy patrí aj CPM, ktorá bola ako súčasť práce implementovaná. Jej popis sa nachádza v sekcii 4.1. Samozrejme globálnych prístupov existuje oveľa viac (*spektrálny rozklad*, rôzne *greedy* metódy, a iné), no ich zhrnutie a popis je mimo rozsah tejto práce. Pre detailný prehľad viď [2].

Ako zhrnutie konštatujeme teda, že globálne metódy je vhodné použiť na siete :

- o ktorých máme kompletný prehľad,
- ktoré sa dynamicky nevyvíjajú,
- ktoré vykazujú hierarchickú štruktúru s predpokladom neprekrývajúcich sa komunít.

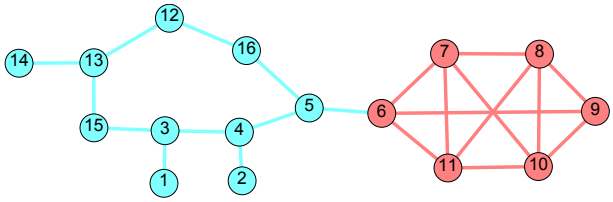
3.3 Lokálne metódy

Lokálne metódy pristupujú k problému detekcie komunít, oproti globálnym, „z opačného konca“. Nepotrebujú poznať štruktúru celej siete. Vo všeobecnosti tieto metódy začínajú od určitého vrcholu, alebo s nejakou množinou *počiatočných skupín* (z angl. *seed groups*), ktoré sú postupne optimalizované vzhľadom, na určitú lokálne definovanú funkciu *hustoty* (z angl. *density*). Skupina je optimalizovaná pridávaním, alebo odoberaním vrcholov, zo svojho najbližšieho okolia. Skupina sa považuje za komunitu, keď pridaním, alebo odobraním vrcholu nemôžeme vylepšiť jej kvalitu vzhľadom na použitú funkciu hustoty.

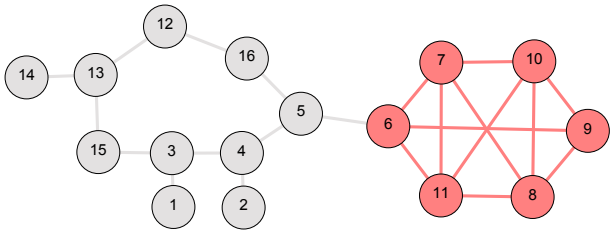
Aj napriek tomu, že metód pracujúcich na princípe lokálnej optimalizácie existuje veľa, všetky sa viac-menej zhodujú na základnom tvare funkcie hustoty $d(S)$, ktorá je použitá pri optimalizácii *seed* skupín. Vyjadrená je ako

$$d(S) = \frac{w_{in}}{w_{in} + w_{out}} \tag{7}$$

kde w_{in} je počet hrán *interných* pre skupinu S a w_{out} je počet hrán, ktoré spájajú skupinu S so zvyškom siete. Táto funkcia je často modifikovaná, lebo vo svojom základnom tvare nemusí zachytávať také komunity, aké hľadáme. Uvedená funkcia a všetky jej variácie



Obr. 12: Motivácia použitia lokálnych metód (CIS).



Obr. 13: Problém detekcie riedkej komunity s globálnou metódou (CPM).

vyjadrujú v podstate lokálnu mieru modularity s cieľom minimalizovať externé a maximalizovať interné hrany komunity [8][9].

Dôležité je si uvedomiť, že lokálne komunity, sú vytvárané **len** vzhľadom na svoje najbližšie okolie (susedné vrcholy). To nám dovoľuje odhaliť široké spektrum komunít (rôznej štruktúry), či už s malými hustotami, alebo veľkými. Takáto situácia je znázornená na obr. 12, kde sme použili lokálnu metódu CIS 4.2, ktorá odhalila 1 hustú komunitu a 1 riedku, kruhovú komunitu. Odhalenie kruhovej komunity sa však nemusí podariť vždy, viď sekcia 4.2.3. Globálne metódy by mali v tomto prípade problém detekovať riedky zhluk 4.1, viď obr.13, kde bola použitá metóda CPM.

S lokálnymi metódami je spojený pojem *pokrytie grafu* z (angl. *cover of graph*).

Definícia 3.3 *Pokrytie grafu G predstavuje takú množinu komunít, pričom každý vrchol grafu G patrí aspoň do jednej z týchto komunít [2].*

Základy, o ktoré sa opierajú lokálne metódy môžeme zhrnúť nasledovne :

- Komunita je *lokálne* definovaný objekt.
- Komunita na „jednom konci“ siete, by nemala byť ovplyvnená tým, čo sa deje na „druhom konci“ siete.
- Štruktúra komunít sa môže v rôznych častiach siete líšiť - niektoré môžu byť spojené husto, iné riedko.

Ďalej uvádzame prehľad niektorých *lokálnych metód*, ktoré detekujú *prekrývajúce sa komunity*. Čerpali sme z [9] a článkov jednotlivých algoritmov.

3.3.1 Rank Removal, Iterative Scan - RaReIS

V [10] bol publikovaný lokálny prístup založený na dvoch fázach (ďalej označovaný, podľa svojich fáz, ako *RaReIS*). V prvej fáze je použitý algoritmus *Rank Removal* (*RaRe*).

3.3.1.1 Rankremoval fáza - RaRe RaRe je založený na myšlienke rozkladu grafu na menšie spojité komponenty, čo sa dosahuje odstraňovaním „vysoko ohodnotených" vrcholov. Ako ohodnotenie vrcholov môžeme použiť napr. Page-Rank [27], alebo samotný stupeň vrcholu. Chceme odstrániť také vrcholy, ktoré graf rozpoja čo najviac. Odstránené vrcholy sú pridané do množiny R. Proces odstraňovania sa opakuje, pokiaľ sa nedosiahne veľkosti komponent v určitom, zadanom rozsahu. Každá z komponent sa dá považovať za *jadro* (z angl. *core*) nejakej komunity. Ďalej sa pokračuje tak, že vrcholy z R sa pridávajú k týmto vytvoreným jadrám. Ak je vrchol z R pridaný k viac ako jednému klastru, potom sa tieto klastre *prekrývajú*. Treba si však uvedomiť, že jadrá sa neprekrývajú a komunikujú spolu len cez vrcholy v R. Vrcholy sú pridané k jadrám s ktorými sú *susedné* a k jadrám, pri ktorých dôjde ich pridaním, k navýšeniu metriky W, ktorá bola v článku použitá v spomínanom tvare 7 zo str. 24. Pseudokódy jednotlivých častí RaRe sú uvedené ďalej.

Pseudokód 2 RaRe(*G,W*)

Require: W - metrika ohodnotenia (density funkcia)
  global $R \leftarrow \emptyset$
  $\{H_i\}$ sú spojité komponenty v G
  for all H_i **do**
    ClusterComponent(H_i); - *pseudokód procedúry ClusterComponent je zobrazený nižšie*
  end for
  Počiatočné klastre $\{C_i\}$ sú jadrá komunít.
  for all $v \in R$ **do**
    for all Klastre $\{C_i\}$ **do**
      Pridaj v do klastra C_i, ak v je susedom C_i alebo $W(v \cup C_i) > W(C_i)$;
    end for
  end for

Druhá fáza RaReIS je *iteratívny sken* IS (z angl. iterative scan).

3.3.1.2 Iterative scan - IS Algoritmus IS rozširuje klastre, ktoré pripravil RaRe v prvej fázy - slúžia ako *seed* skupiny pre IS. Rozširovanie prebieha pomocou pridávania alebo odoberania jedného vrcholu v danom čase. Podmienkou pridania, resp. odobrania je zlepšenie metriky W. IS končí ak sa klaster už nedá zlepšiť. Keďže sa jedná o heuristický prístup, tak výsledná podoba klastru závisí na poradí spracúvaných vrcholov a poskytnutých *seed* skupinách.

Počas behu IS sa klaster C postupne mení, vzniká sekvencia zmien $C_1, C_2, ...$, s tým, že je dodržané, že $W(C_1) < W(C_2) < ....$ Nerovnosti sú ostré, čo znamená, že sa vyhneme

Pseudokód 3 ClusterComponent(H)

Require: t - počet vrcholov, ktoré majú byť odstránené; min, max - min., max. veľkosť jadra
  if $|V(H)| > max$ **then**
    $\{v_i\}$ je *t* najvyššie ohodnotených vrcholov v H;
    $R \leftarrow R \cup \{v_i\}$; $H \leftarrow H \backslash \{v_i\}$;
    $\{F_i\}$ sú spojité komponenty v H;
    for all F_i **do**
      ClusterComponent(F_i);
    end for
  else if $min \leq |V(H)| \leq max$ **then**
    označ H ako jadro komunity;
  end if

opakovaniu rovnakého C v sekvencii. Keďže je počet možných klastrov konečný, tak IS musí skončiť pri poskytnutí akýchkoľvek *seed* skupín. Pseudokó IS je uvedený nižšie.

Pseudokód 4 IterativeScan(seed,*G*, *W*)

Require: $seed$ -počiatočné seed skupiny; G - skúmaný graf; W - metrika ohodnotenia (density funkcia)
  $C \leftarrow seed$; $w \leftarrow W(C)$;
  increased $\leftarrow$ TRUE;
  while increased **do**
    for all $v \in V(G)$ **do**
      if $v \in C$ **then**
        $C' \leftarrow C \backslash \{v\}$;
      else
        $C' \leftarrow C \cup \{v\}$;
      end if
      if $W(C') > W(C)$ **then**
        $C \leftarrow C'$;
      end if
    end for
    if $W(C) = w$ **then**
      $increased \leftarrow FALSE$;
    else
      $w \leftarrow W(C)$;
    end if
  end while
  return C;

Problémom pri IS je to, že počas behu algoritmu môžu vzniknúť rozpojené klastre (viď popis *CIS*). Tento problém dal za vznik algoritmu *CIS* (sek. 4.2, str. 38).

3.3.2 Lancichinetti-Fortunato method - LFM

LFM [12] pristupuje k detekcii komunít, spomínaným, typickým lokálnym prístupom z úvodu sekcie 3.3. *Seed* skupinu C predstavuje náhodne vybraný vrchol v. Skupina C je optimalizovaná vzhľadom na upravenú funkciu hustoty, definovanú ako

$$f(C) = \frac{k_{in}^C}{(k_{in}^C + k_{out}^C)^\alpha}, \tag{8}$$

kde k_{in}^C a k_{out}^C je celkový počet vnútorných a vonkajších stupňov vrcholov skupiny C. Parameter α je kladné reálne číslo, pomocou ktorého kontrolujeme veľkosť hľadaných komunít.

Autori vo funkcii hustoty, urobili okrem parametra α, ešte modifikáciu spojenú s určením hodnoty k_{in}^C. Počet vnútorných hrán násobia 2, takže vzhľadom na rovnicu 7 zo str. 24 dostávame

$$f(C) = \frac{w_{in} * 2}{(w_{in} * 2 + w_{out})^\alpha}. \tag{9}$$

Podobne modifikovaná funkcia hustoty je použitá aj v metóde CIS 4.2, str. 38.

Autori ďalej pomenovali komunitu, ktorá vznikne rozširovaním počiatočného vrcholu v, ako *prirodzenú komunitu* vrcholu v (z angl. *natural community*). Zavádzajú tiež koncept ohodnotenia vrcholu, na základe jeho „vhodnosti“ priradenia do komunity (z angl. *node fitness*).[2] *Fitness* vrcholu v vzhľadom na skupinu C môžeme určiť pomocou *fitness funkcie* danej v tvare

$$f_C^v = f_{C+\{v\}} - f_{C-\{v\}}, \tag{10}$$

kde $f_{C+\{v\}}$ a $f_{C-\{v\}}$ označuje „prítomnosť“ a „neprítomnosť“ vrcholu v v skupine C.

Prirodzenú komunitu vrcholu v detekuje LFM nasledujúcim postupom :

1. prejdeme všetky susedné vrcholy C, nepatriace do C
2. sused s najväčšou *fitness* je pridaný do C (C je modifikované)
3. prepočíta sa *fitness* každého vrcholu C
4. ak sa nájde vrchol s negatívnou *fitness*, tak je odstránený z C (C je modifikované)
5. ak nastane 4, tak opakuj od 3, ináč opakuj od 1 pre modifikované C

[2] Čitateľovi sa ospravedlňujem, za „kostrbatý“ preklad pojmu *node fitness*, no nenapadlo ma nič vhodnejšie.

Proces končí, keď majú všetky vrcholy v 1. kroku negatívnu *fitness*. Všimnime si, že sa jedná o greedy prístup. Za účelom detekovania *krytu* grafu (viď 3.3, str. 25) by sme mohli spustiť vyššie uvedenú procedúru z každého vrcholu grafu C. Vzhľadom na to, že sú ale *prirodzené komunity* mnohých vrcholov rovnaké, väčšina výpočtového času by sa strávila znovu-odhaľovaním rovnakých komunít. Autori preto zvolili nasledujúci postup :

1. zvoľ náhodný vrchol v
2. vyhľadaj prirodzenú komunitu vrcholu v
3. náhodne vyber vrchol s, ktorý nie je priradený do žiadnej komunity
4. vyhľadaj prirodzenú komunitu vrcholu s, prechádzajúc všetky vrcholy, bez ohľadu na ich možné členstvo v nejakej komunite
5. opakuj od 3.

Algoritmus končí, keď sú všetky vrcholy priradené do nejakej komunity. Zdôvodnenie tohto postupu je nasledujúce: Komunita C bola odhalená na základe rozširovania okolo nejakého vrcholu. Ak vyberieme akýkoľvek iný vrchol z tejto komunity C, tak odhalíme buď rovnakú komunitu C, alebo nejakú s ktorou sa prekrýva. Prekrývajúcu komunitu však nájdeme aj keď začneme od vrcholu mimo komunity C. Prekrývajúce sa vrcholy budú zahrnuté počas vytvárania komunity vďaka tomu, že počas detekcie v kroku č. 4 neberieme do úvahy členstvo vrcholu v nejakej komunite. Autori v článku popisujú, že na základe vykonaných testov je strata presnosti pri použití tohto postupu minimálna.

Dôležitou súčasťou výpočtu hodnoty *fitness* vrcholu, je rezolučný parameter α. S jeho pomocou môžeme ovplyvňovať „pohľad“ na sieť. Veľké hodnoty tohto parametra majú za následok nájdenie malých komunít a naopak. Autori zistili, že pre $\alpha < 0.5$ vzniká väčšinou 1 komunita a pre $\alpha > 2$ vznikajú najmenšie komunity. Prirodzenou voľbou je teda $\alpha = 1$, čo korešponduje s definíciou tzv. *slabej komunity* (z angl. *weak community*) uvedenou v [13]. Analýza siete pre rôzne hodnoty α, nám poskytuje informácie o *stabilných* komunitách (také, ktoré sa nachádzajú v *krytu* grafu pre rôzne α).

LFM silne závisí na hodnote parametra α. Časová zložitosť pre fixnú veľkosť α je zhruba $O(n_c s^2)$, kde n_c je počet komunít a s je priemerná veľkosť komunity. V najhoršom prípade je zložitosť $O(n^2)$.

3.3.2.1 Merging of overlapping natural communities - MONC

MONC [16] využíva upravenú verziu fitness funkcie z LFM

$$f(c) = \frac{k_{in}^c + 1}{(k_{in}^c + k_{out}^c)^\alpha}, \tag{11}$$

ktorá dovoľuje, aby bol samostatný vrchol považovaný za komunitu. Týmto sa vyhneme porušeniu princípu *lokálnosti*. Uvedená fitness funkcia umožňuje, aby MONC našiel hodnoty α (rezolučný parameter, ako v prípade LFM), pre ktoré je skupina vrcholov lokálne optimálna. Namiesto numerického zisťovanie týchto α hodnôt, MONC vypočíta

nasledujúcu najmenšiu hodnotu α, ktorá spôsobí ďalšie rozšírenie a pokračuje v rozširovaní komunity. V prípade, že prirodzená komunita vrcholu i je podmnožina iného vrcholu, tak analýza končí. Týmto spôsobom, MONC zlučuje komunity počas spracúvania s výsledkom rýchlejšieho preskúmania siete oproti LFM.

3.3.3 Agglomerative hierarchical clustering based on maximal clique - EAGLE

Na začiatok treba poznamenať, že EAGLE [19] v prvej fáze vyhľadáva maximálne kliky v grafe (podobne ako CPM, 4.1, str. 34). Kvôli tomu ho môžeme považovať za globálny algoritmus. Na druhú stranu však ďalej využíva princíp lokálnej expanzie (čo je jeho hlavná devíza), ktorý je charakteristický pre lokálne algoritmy. Preto ho uvádzame v tejto sekcii.

Rovnako, ako pre mnohé iné lokálne algoritmy, aj pre EAGLE je dôležité, aby pracoval s vhodnými *seed* skupinami, ktoré budú rozširované. Autori EAGLE sa rozhodli zvoliť za *seed* skupiny množinu *klík* (viď 2.3 a 2.4, str. 10). Klika má zo všetkých typov podgrafov najhustejšie prepojené vrcholy. Autori preto kliku považujú za vhodného kandidáta, na vytvorenie jadra potenciálnej komunity.

Na vyhľadanie maximálnych klík grafu bol v článku použitý algoritmus *Bron-Kerbosch* (viď 4.1.2, str. 37). Maximálne kliky, ktoré obsahujú vrcholy ktoré patria zároveň do inej, väčšej, max. kliky sa nazývajú *podriadené/závislé max. kliky* (z angl. *subordinate maximal cliques*). EAGLE s takýmito klikami nepracuje. Väčšina závislých klík má malú veľkosť. V EAGLE sa preto odstránia všetky kliky, ktoré majú menšiu veľkosť ako nejaký threshold k (zvyčajne v rozpätí 3-6). Po tomto odstránení sa môže stať, že niektoré vrcholy nepatria do žiadnej kliky. Tieto vrcholy, potom označujeme analogicky, ako *podriadené/závislé vrcholy* (z angl. *subordinate vertices*).

EAGLE pracuje, podobne ako RaReIS (sek. 3.3.1) v dvoch fázach. V prvej fáze sa vygeneruje dendogram (viď obr.11 na str. 22). V druhej fáze sa zvolí vhodný *rez* dendogramu, čím sa dendogram rozpadne na komunity.

Prvá fáza algoritmu môže byť opísaná nasledovne :

1. Nájdi všetky maximálne kliky v danej sieti. „Zahoď" *závislé kliky*. Tie, ktoré ostanú považuj za počiatočné *core* komunity. Každý *závislý vrchol* považuj taktiež, za počiatočnú *core* komunitu. Spočítaj *podobnosť* M medzi každým párom komunít.

2. Vyber pár komunít s najväčšou podobnosťou, zlúč ich do jednej komunity a spočítaj M, medzi touto novo vytvorenou komunitou a ostatnými komunitami.

3. Opakuj 2. krok, pokiaľ neostane len 1 komunita.

V algoritme je podobnosť M medzi dvoma komunitami C_1 a C_2 definovaná ako

$$M = \frac{1}{2m} \sum_{v \in C, w \in C_2, v \neq w} \left[A_{vw} - \frac{k_v k_w}{2m} \right]. \tag{12}$$

A_{vw} je prvok matice susednosti (viď 2.1.6.2, str. 14) siete (pracuje sa s neohodnotenými, neorientovanými sieťami). $m = \frac{1}{2} \sum_{vw} A_{vw}$ je celkový počet hrán v sieti a k_v je stupeň vrcholu v.

Druhá fáza algoritmu má za úlohu rozdeliť dendogram. Ako už bolo spomenuté v popise globálnych metód, na rozhodnutie, kde dendogram rozdeliť, je použitá miera *modularita* (6, str. 23. EAGLE používa upravenú verziu modularity EQ. Ak máme daný *cover* siete, tak O_v označuje počet komunít, do ktorých patrí vrchol v. Upravená verzia modularity je vyjadrená nasledovne

$$EQ = \frac{1}{2m} \sum_i \sum_{v \in C_i, w \in C_i} \frac{1}{O_v O_w} \left[A_{vw} - \frac{k_v k_w}{2m} \right]. \tag{13}$$

Z EQ sa stáva Q, ak každý vrchol patrí len do jednej komunity. EQ = 0, keď všetky vrcholy patria do rovnakej komunity.

Aj keď odmyslíme čas potrebný na nájdenie všetkých klík, tak je EAGLE výpočtovo náročný algoritmus, so zložitosťou $O(n_2 + (h + n)s)$, kde s je počet maximálnych klík a h je počet párov max. klík, ktoré sú susedné.

3.3.3.1 Greedy clique expansion - GCE Podobne ako EAGLE, tak aj GCE [20] používa kliky, ako *seed* skupiny. Rozširuje ich však, na základe greedy optimalizácie lokálnej *fitness funckie*. GCE odstraňuje komunity, ktoré sú podobné komunitám, ktoré už boli objavené. To za pomoci stanovenia *vzdialenosti* medzi dvoma komunitami C_1 a C_2 definovanej ako

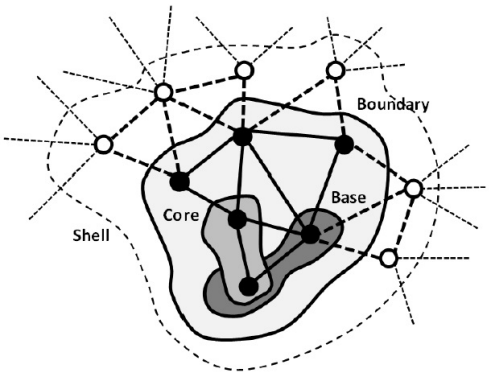
$$1 - \frac{|C_1 \cap C_2|}{min(|C_1|, |C_2|)}. \tag{14}$$

Ak je táto vzdialenosť menšia, ako daný parameter ϵ, tak sú si komunity podobné. Časová zložitosť pre GCE je $O(mh)$, kde m je počet hrán a h je počet klík.

3.3.4 OSLOM

OSLOM (Order Statistics Local Optimization Method) [17] testuje štatistickú dôležitosť klastra vzhľadom na globálny *null model* (napr. vygenerovaný *náhodný graf*, konfiguračným modelom [18]) počas rozširovania komunity. Za účelom rozšírenia komunity, sa pre každý susedný vrchol stávajúcej komunity vypočíta r hodnota. r je kumulatívna pravdepodobnosť toho, že vrchol má počet vnútorných hrán (vzhľadom na komunitu) rovný, alebo väčší, ako je počet hrán zo suseda, do tejto komunity v *null modele*. Ak je kumulatívna distribúcia najmenšej hodnoty r menšia, ako nejaká daná tolerancia, tak sa r považuje za dôležitú a korešpondujúci vrchol je pridaný do komunity. Inak sa vyhodnotí druhé najmenšie r, atď. V jednoduchosti sa dá povedať, že vrchol i je do komunity C priradený, pokiaľ s ňou zdieľa oveľa viac hrán, ako sa očakávalo vzhľadom na *null model* (vzťah i s C je „nečakane silný“).

OSLOM zvyčajne identifikuje veľké množstvo jednočlenných a odľahlých komunít. Náročnosť v najhoršom prípade je $O(n^2)$, pričom konkrétna náročnosť závisí na tom, akú majú komunity v skúmanej sieti štruktúru.



Obr. 14: Znázornenie pojmov použitých pri popise lokálnych metód.

3.3.5 Local community detection based on vertex dependency - VDM

Popíšeme len v stručnosti, pre viac viď [14]. Táto metóda pracuje s nasledujúcimi pojmami (dajú sa aplikovať aj na ostatné lokálne metódy) :

- jadro komunity - C (*community core*) - vrcholy, ktorých všetky hrany patria do C
- hranica komunity - B (*community boundary*) - vrcholy, ktoré majú aspoň jednu hranu, ktorá nepatrí do C (majú suseda v S)
- plášť komunity - S (*community shell*) - vrcholy, ktoré nie sú spojené s C
- základ komunity (*community base*) - vrchol, alebo množina vrcholov, z ktorých sú generované komunity (u iných metód označené ako *seed* skupiny)

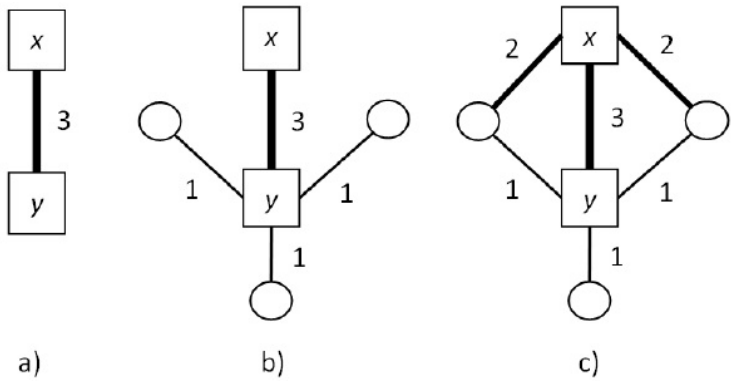
Graficky sú pojmy znázornené na obr. 14 [15].

Táto metóda je založená (tak ako ostatné lokálne metódy) na iteratívnej expanzii *základu komunity* (*seed/base*). Pracuje s ohodnotenými neorientovanými grafmi. Ako kritérium na pridanie vrcholu do komunity sa používa miera *závislosť vrcholu* (*vertex dependency*). Miera sa snaží zachytiť vzťah medzi dvoma vrcholmi, resp. medzi vrcholom a množinou vrcholov. Koncept závislosti vrcholu autori vysvetľujú pomocou obr. 15 [15].

Na obr. 15*a)* vidíme, že vzťah medzi vrcholmi x a y je vyvážený, pretože zdieľajú len 1 hranu. V *b)* je však vrchol y menej závislý na vrcholu x, ako opačne. V *c)* vidíme, že x už nie je, vďaka 2 novým hranám, tak závislý na y. V *c)* treba brať do úvahy aj to, že nové hrany spájajú x so susedmi y, čím sprostredkúvajú časť závislosti x na y. Pre formálny zápis a detailnejšie vysvetlenie viď [14].

Pseudokód algoritmu je uvedený na str. 33. Výsledkom je komunita L, $L = C \cap B$.

V ďalšej sekcii podrobnejšie rozoberieme metódy CIS a CPM, ktoré boli ako súčasť práce implementované.



Obr. 15: Ukážka závislosti medzi dvoma vrcholmi.

Pseudokód 5 Vertex dependency method(G, n_0, C, S, B)

Require: sieť G, počiatočný vrchol n_0
Require: prázdne jadro komunity C, prázdny plášť komunity S
Require: prázdne ohraničenie komunity B, čo sa rovná *základu komunity*
Pridaj n_0 do B, pridaj všetkých susedov n_0 do S
while najmenej 1 vrchol z S bol rozpoznaný **do**
1. Presuň vrcholy z ohraničenia B, ktoré nemajú susedov mino komunity L, do jadra C
2. Naplň plášť S novými susedmi vrcholov, ktoré boli pridané do B a sú mimo komunity L
3. Vypočítaj závislosť na ostatné vrcholy, pre každý vrchol plášťa S
4. Presuň do ohraničenia B každý vrchol z plášťa S, ktorý spĺňa kritéria pre rozpoznanie (viď v [14])
5. Vytvor *community closure* (viď v [14]) z ostávajúcich vrcholov plášťa a naplň plášť S s novými susedmi vrcholov, ktoré boli pridané do *closure*.
end while
for all vrcholy zo S **do**
1. Presuň vrchol do *community closure*, ak spĺňa kritériá
2. Naplň plášť S novými susedmi vrcholov, ktoré boli pridané do closure
end for
return community L, $L = C \cap B$

4 CIS a CPM

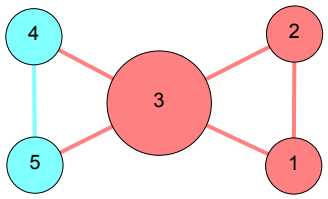
Ako súčasť práce sme sa rozhodli implementovať metódy CIS [8] a CPM [3], ktoré budú detailne popísané nižšie. Metódu *perkolácie klík* (CPM) sme vybrali ako „referenčnú metódu“, pretože generuje prekrývajúce sa komunity, ktoré sú okrem toho klikami, tj. akýsi ideál. CPM je zaujímavá v tom, že k tomu, aby sme ju mohli použiť síce potrebujeme poznať celý graf, ale vo svojej podstate odhaľuje špecifické štruktúry na lokálnej úrovni, ktoré nie sú ovplyvnené zvyškom grafu (dá sa teda považovať v určitom zmysle aj za lokálnu metódu). Výsledky CPM sme chceli preto porovnať s nejakou čisto lokálnou metódou. Ako reprezentanta lokálnych metód sme vybrali metódu *iteratívneho scanu s kontrolou konektivity* (CIS). V prípade CIS sme implementovali aj niekoľko modifikacií (viď 5.4), oproti verzii popísanej v článku [8].

4.1 Clique percolation method - CPM

Táto metóda bola prezentovaná v článku [3]. Podobne ako pri metóde EAGLE (3.3.3, str. 30) autori aj tu považujú za vhodné jadrá komunít *kliky* (2.3, str. 10). Opierajú sa o pozorovanie, že typická komunita pozostáva z niekoľkých kompletných podgrafov. Zavádzajú pojem *k-clique-community* („komunita k-klík“), ktorú definujú ako zjednotenie všetkých *k-klík* (2.5, str. 10), ktoré sú vzájomne dosiahnuteľné cez sériu *susedných k-klík* (z angl. *adjacent k-cliques*). Dve k-kliky sú susedné, ak zdieľajú aspoň *k-1* vrcholov. Vďaka tejto definíci susednosti dokáže CPM odhaliť prekrývajúce sa komunity.

Napr.: 2 k-kliky k_1 a k_2 sa kryjú na určitých vrcholoch, no tých je menej ako $n-1$. k_1 preto nie je dosiahnuteľná z k_2 a opačne. Každá z týchto 2 k-klík je teda zaradená do samostatnej komunity, s tým že sa prekrývajú na zdieľaných vrcholoch.

Táto situácia je pre názornosť ukázaná na obr. 16, kde sú znázornené dve 3-kliky, $k_1 = \{1, 2, 3\}$, $k_2 = \{3, 4, 5\}$. Kliky nie sú vzájomne dosiahnuteľné ($1 < k - 1$), pričom predstavujú komunity, ktoré zdieľajú vrchol č.3.



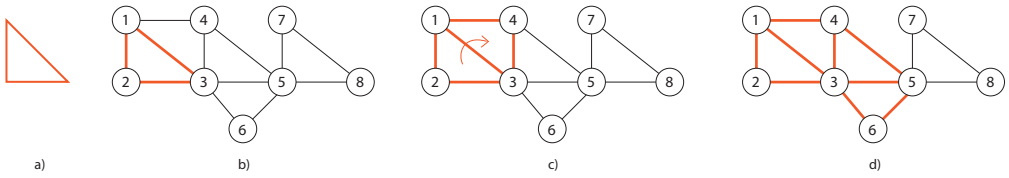
Obr. 16: Ukážka dvoch vzájomne nedosiahnuteľných 3-klík.

Keď zhrnieme vyššie uvedené, tak platí, že : *Konkrétna k-klika môže patriť len do jednej k-clique-komunity, ale k-kliky z rôznych komunít môžu zdieľať vrcholy.*

To aké komunity nájdeme silne závisí od parametra k. Zväčšovaním k sa sieť viac rozpadá a dostávame menšie komunity, ktoré sú ale viac prepojené.

Pre predstavu získania k-clique-komunít môže byť použité presúvanie akéhosi „k-clique vzoru“. K-clique vzor vyzerá ako kompletný graf o k vrcholoch. Takýto vzor po-

tom môže byť „položený“ na akúkoľvek k-kliku grafu a „posunutý“ ďalej na susednú k-kliku tak, že presunieme jeden vrchol vzoru a zvyšné vrcholy (*k-1*) ostanú fixné (vzor ako keby preklápame cez 1 hranu). Ilustrácia je na obr. 17.



Obr. 17: a) $k = 3$ vzor; b) počiatočná pozícia; c) presun z *b)*; d) nájdená k-komunita

K-clique-komunity grafu sú potom všetky podgrafy, ktoré môžu byť celé preskúmané pomocou presúvania spomenutého vzoru - vzor sa nesmie dostať „mimo“ daného podgrafu. Pre $k = 2$ odhalíme spojité komponenty grafu.

Detekcia všetkých k-klík by bola časovo extrémne náročná. Keď však uvážime, že kompletný podgraf o veľkosti s je možné vytvoriť kombináciou rovnakých k-klík o veľkosti $k \leq s$, tak je lepšou stratégiou vyhľadať maximálne kompletné podgrafy siete [4]. Potom už len stačí analyzovať prekrytia medzi nimi a vyhľadať v nich k-clique-komunity.

4.1.1 Postup algoritmu CPM

Algoritmus sa začína detekciou všetkých maximálnych kompletných podgrafov (max. klík) v skúmanej sieti. Z praktického uhla pohľadu je vyhľadávanie max. klík NP-kompletný problém [5]. Metóda *Bron-Kerbosch* pracuje v čase rastúcom exponenciálne s veľkosťou grafu. Detekcia teda prebieha pomocou algoritmu *Bron-Kerbosch*, ktorý je bližšie opísaný v sek. 4.1.2.

Po tom, čo sme našli všetky max. kliky pripravíme *clique-clique overlap matrix* (*matica prekrytia max. klík*) O. Jedná sa o symetrickú maticu o veľkosti $n_c \times n_c$, kde n_c je počet max. klík. Každý riadok a stĺpec reprezentuje nájdenú max. kliku. Prvky matice predstavujú počet spoločných vrcholov medzi danými dvoma max. klikami. Diagonálne prvky sa rovnajú veľkosti max. kliky v danom riadku/stĺpci.

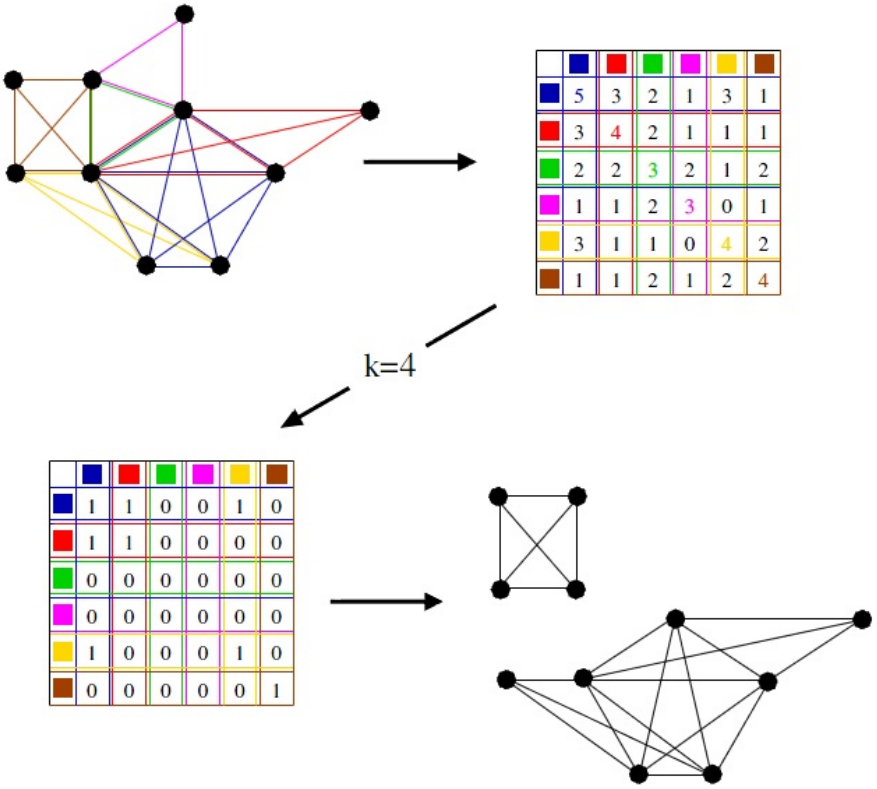
k-clique-komunity pre danú hodnotu k sú ekvivalentné takým susedným max. klikám, ktoré sú vzájomne prepojené aspoň $k - 1$ vrcholmi a zároveň samotné musia mať aspoň k vrcholov.

Z *matice prekrytia max. klík* potom získame k-komunity nasledovne :

1. Každý prvok mimo diagonály, ktorý je menší ako $k - 1$ nahradíme 0
2. Každý diagonálny prvok menší ako k nahradíme 0
3. Ostatné prvky nahradíme 1

4. Spravíme komponent analýzu matice - Pre každý riadok zlučujeme nenulové prvky (max. kliky), čím pre tento riadok vytvárame k-clique-komunitu. Po každom spracovanom riadku skontrolujeme, či vytvorená k-clique-komunita nemá nejakú max. kliku spoločnú s už vytvorenou k-clique-komunitou. Ak áno, tak ich zlúčime.

Autori uviedli v [4] pekný názorný príklad, viď obr. 18. Na obr. sú v počiatočnom grafe sú farebne znázornené max. kliky. Z nich zostrojíme *maticu prekrytia klík*. Tú potom redukujeme podľa postupu (získavame $k = 4$ k-komunity). Po redukcii maticu spracujeme a dostávame výsledné k-komunity.

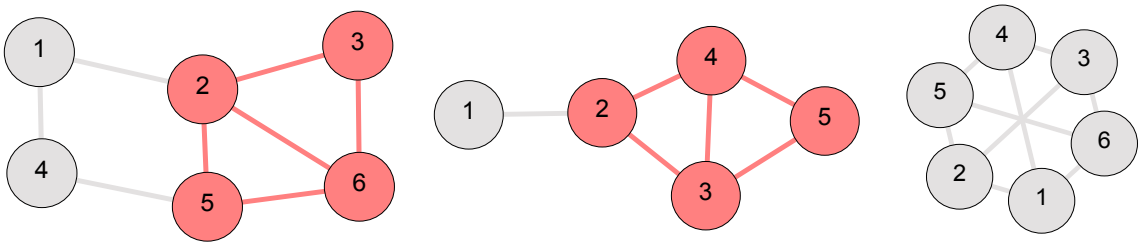


Obr. 18: Postup CPM.

Testami autori zistili, že najlepšie komunity dostaneme pre hodnoty $3 \leq k \leq 6$. To platí hlavne pre analýzu sociálnych sietí.

CPM má u mnohých reálnych sietí polynomálnu časovú zložitosť. Silne však závisí na vstupných dátach, hlavne čo sa týka hustoty grafu. Preto často zlyháva u veľkých sietí (cca. 100 tis. vrcholov a viac) [4][9]. Vo všeobecnosti sa taktiež pre neznámu sieť ťažko odhaduje, akú hodnotu k máme zvoliť, aby sme dostali „rozumné komunity“. Túto metódu je kvôli jej definícii komunity, ktorá sa opiera o kliky (čím striktne určuje kompletné vzájomné prepojenie všetkých vrcholov komunity), vhodnejšie použiť pre grafy, ktoré

obsahujú husto prepojené časti. Na druhú stranu, ak graf obsahuje veľké množstvo klík, metóda môže vrátiť triviálnu komunitu obsahujúcu celý graf. Veľkým problémom je to, že CPM nezaradí do komunít veľké množstvo vrcholov, ako napr. *listy* grafu, alebo husto prepojené podgrafy, ktorým chýba 1 hrana, aby splnili kritérium kliky. Príklady takýchto problémových situácií sú na obr. 19. Na obr. šedé vrcholy neboli zaradené do žiadnej komunity. Hlavne na 3. grafe sa ukazuje, že aj keď je husto prepojený CPM kvôli absencii kliky nedetekuje žiadnu komunitu.



Obr. 19: Ukážky grafov, s ktorými má CPM problémy.

Ďalej uvádzame algoritmus *Bron-Kerbosch*, ktorým získame z grafu všetky max. kliky.

4.1.2 Bron-Kerbosch algoritmus

V priebehu *Bron-Kerbosch* (ďalej *BK*) algoritmu [6] pracujeme s tromi disjunktnými množinami vrcholov :

- K (*Kandidáti*) – skladá sa z vrcholov, ktoré sú spojené so všetkými vrcholami v R, takže ich použitím môžeme kliku R rozšíriť
- R (*Rozširovaná klika*) – patria sem vrcholy, ktoré tvoria aktuálne rozširovanú kliku
- S (*Spracované*) – obsahuje vrcholy, ktoré už boli spracované, alebo inak, ktoré sa už nachádzali v množine K a tým pádom všetky kliky obsahujúce tieto vrcholy už boli nájdené

Dôležité je, že všetky vrcholy, ktoré sú spojené s každým vrcholom v R sa nachádzajú buď v K alebo v S. Úlohou množiny S je vyhnúť sa zisťovaniu rovnakej max. kliky viac krát, cez aktualizáciu $K = K^{\smile}\{v_i\}$ (kde v je aktuálne spracúvaný vrchol). Aby sme sa vyhli vráteniu kliky, ktorá nie je maximálna, tak algoritmus kontroluje, či je množina S prázdna – ak S nie je prázdna, tak vrcholy v S môžu byť pridané do R, ale tým by sme dostali kliku, ktorú sme už predtým našli.

Ak sú pri výpočte množiny S a K prázdne, tak už neexistujú žiadne vrcholy, ktoré by mohli byť pridané do R a teda vrátime R ako maximálnu kliku.

Výpočet je založený na rekurzívnom backtrackingu. Postup *BK* v základnej variante je taký, že najskôr množina K obsahuje všetky vrcholy grafu. Množiny S a R sú prázdne.

Postupne vyberáme vrcholy z K a pridávame ich do R (rekurzívne, aby sme prešli všetky kombinácie presunov). Keď presunieme vrchol v z K do R, tak vrcholy v K, ktoré nie sú susedmi v, sú z K a S odstránené. Potom presunieme v z K do S aby sme ho vylúčili z uváženia pri nasledujúcich max. klikách a pokračujeme ďalej s ďalším vrcholom z K.

Pseudokód 6 BronKerbosch(R, K, S)

```
if K = ∅ and S = ∅ then
  return R
end if
for all v ∈ K do
  R_new = R ∪ {v}
  K_new = K ∩ (susedia{v})
  S_new = S ∩ (susedia{v})
  BronKerbosch(R_new, K_new, S_new)
  K = K\{v}
  S = S ∪ {v}
end for
```

Uvedená štandardná implementácia tohto algoritmu má veľmi zlú výkonnosť v najhoršom prípade. Algoritmus vie, že klika je podgrafom inej max. kliky až vtedy keď je formovanie celej aktuálne zisťovanej kliky v R dokončené. Z toho vyplýva, že sa prejdú všetky kliky, aj tie ktoré nie sú maximálne, čo je zbytočné.

Na odstránenie tohto problému bola do BK pridaná heuristika, ktorá minimalizuje rekurzívne volania [7]. Pracuje sa s *pivotom*. Pivot nám umožní eliminovať duplicitné vetvy rekurzívneho stromu. Pivot (vrchol) sa vyberá z $K \cup S$. Akákoľvek max. klika musí potom obsahovať buď vybratý pivot, alebo niektorý z vrcholov, ktoré nie sú jeho susedmi (inak by mohla byť klika doplnená ešte o vybraný pivot). Tým pádom len pivot a jeho ne-susedia musia byť otestovaní ako možnosti pre vrchol, ktorý má byť pridaný do S (v každom rekurzívnom volaní).

4.2 Connected iterative scan - CIS

Na začiatok je nutné poznamenať, že metóda *CIS* bude v tejto sekcii opísaná tak, ako ju opisujú autori v článku [8]. Na základe tohto opisu sme ju taktiež za účelom porovnania implementovali. V implementačnej časti uvedieme potom nami upravené verzie (5.4.2, str. 55 a 5.4.1, str. 53), ktoré v experimentoch porovnáme s touto pôvodnou.

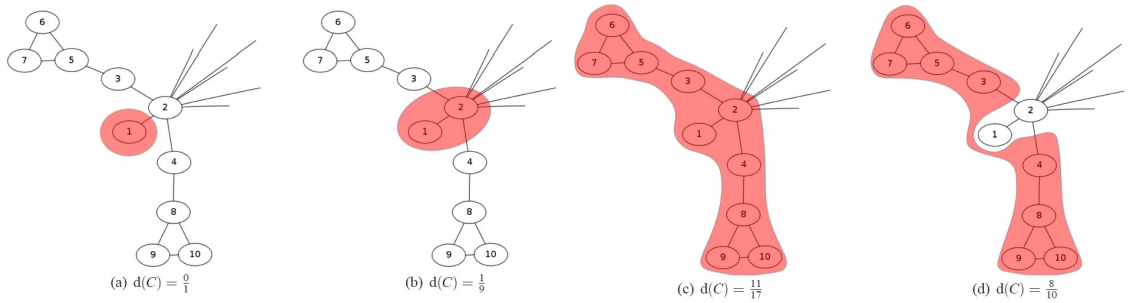
Metóda *CIS* vznikla ako čiastočné vylepšenie metódy *RaReIS* opísanej v sekcii 3.3.1, str. 26. Ako v prípade RaReIS, tak aj v prípade CIS sa jedná o lokálnu metódu. Problémom pri RaReIS bolo, že počas behu algoritmu mohli kvôli odoberaniu vrcholov z rozširovanej *seed* skupiny vzniknúť rozpojené časti. Treba si totiž uvedomiť, že pridávanie/odoberanie vrcholu do/z seed skupiny sa deje na základe jej hustoty a rozdelenia stupňa (vnútorné vs. vonkajšie hrany) vrcholu *v čase pridania/odobrania*. To môže spôsobiť, že vrchol,

Pseudokód 7 BronKerboschPivot(R, K, S)

```
if K = ∅ and S = ∅ then
  return R
end if
vyber pivot p z K ∪ S
for all v ∈ (K\susedia{v}) do
  R_new = R ∪ {v}
  K_new = K ∩ (susedia{v})
  S_new = S ∩ (susedia{v})
  BronKerboschPivot(R_new, K_new, S_new)
  K = K\{v}
  S = S ∪ {v}
end for
```

ktorý bol pridaný v predchádzajúcich krokoch, keď mala seed skupina menšiu hustotu, bude teraz odobraný, čím sa môže skupina rozpojiť.

Príklad takejto situácie je ukázaný na obr. 20 [8]. Na obr. bola použitá na optimalizáciu *seed* skupiny funkcia hustoty $d(S)$ v základnom tvare 7 zo str. 24. Uvažujme seed skupinu tvorenú len vrcholom č. 1. Na začiatku je hustota skupiny $d(S) = 0$, pretože skupina neobsahuje žiadne vnútorné hrany. Po tom, čo prejdeme všetky vrcholy na základe ich *rastúceho stupňa* pripojíme ku skupine vrchol č.2 (hustota sa zvýši kvôli vzniku vnútornej hrany). Postupne sa dostaneme až do stavu c). V tomto bode bude odstránený vrchol č.2, pretože sa tým zväčší hustota skupiny. Vznikajú 2 rozpojené skupiny. Nakoniec sa odstráni vrchol č. 1, čím dostaneme lokálne optimálne rozpojené skupiny, ktoré sa považujú za 1 komunitu.



Obr. 20: Ukážka vzniku rozpojenej komunity.

CIS, tento problém rieši tak, že po každom vykonanom skene (viď 3.3.1.2, str. 26) kontroluje konektivitu nájdenej komunity, z toho aj názov metódy *Connected* Iterative scan. Mimo tejto kontroly pracuje CIS takmer rovnako ako *IS fáza* v RaReIS.

4.2.1 Postup algoritmu CIS

Najskôr stručne zhrnieme ako pracuje IS fáza z RaReIS (tento popis je rovnaký aj pre CIS). IS pozostáva z opakujúcich sa „skenov", ktoré sa snažia zlepšiť poskytnuté *seed* skupiny. Pre každú seed skupinu sa teda spustí séria skenov, pričom každý nasledujúci sken pracuje so skupinou, ktorú vyvinul sken pred ním. Počas jedného skenu sa vyhodnocuje *každý vrchol grafu* práve raz, buď sa k skupine pridá, alebo odoberie, pokiaľ takáto akcia zvýši hustotu spracúvanej skupiny. Skeny sa opakujú kým sa skupina zlepšuje vzhľadom na danú funkciu hustoty. Ak sa už nedá vylepšiť považuje sa skupina za komunitu. Počas skenu sa vrcholy grafu vyhodnocujú podľa stupňa, od najmenšieho po najväčší.

V prípade CIS sa autori rozhodli priradiť každý vrchol grafu do samostatnej seed skupiny. Ako už bolo spomenuté, po každom dokončenom skene sa v CIS vyhodnocuje konektivita rozširovanej skupiny. Ak skupina pozostáva z viacerých spojitých komponentov, tak sa nahradí za ten komponent, ktorý má najväčšiu hustotu a pokračuje sa v skenoch. Výberom komponentu s najväčšou hustotou sa vyhneme opakovanému optimalizovaniu k rovnakým rozpojeným komponentom.

V reálnych sociálnych sieťach existuje veľké množstvo potenciálnych kandidátov na komunity, preto keď sa skeny ukončia je finálne skupiny vhodné prefiltrovať. Táto filtrácia by sa mala vykonávať na základe charakteru skúmanej siete a toho, aký typ komunít chceme odhaliť. V článku autori uvažujú len skupiny, ktoré majú hustotu $d(S) > \frac{1}{3}$. Tento filter je konzistentný s definíciou *slabej komunity* (3.2, str. 21). Oproti RaReIS je použitá v CIS iná verzia funkcie hustoty $d(S)$. Jej charakteristika je opísaná ďalej v sek. 4.2.2.

Pseudokód CIS :

Pseudokód 8 Connected Iterative Scan

Require: $G = (V, E), S \neq \emptyset$
Ensure: $d(S) \geq d(S \cup \{v\})$ **and** $d(S) \geq d(S \backslash \{v\}), \forall v \in V$

```
zlepsena ← true
while zlepsena == true do
  zlepsena ← false
  for all v ∈ V do
    if v ∈ S then
      if d(S\{v}) > d(S) then
        S ← S\{v}
        zlepsena ← true
      end if
    else
      if d(S ∪ {v}) > d(S) then
        S ← S ∪ {v}
        zlepsena ← true
      end if
    end if
  end forS ← maxKomponent(S)
end while
```

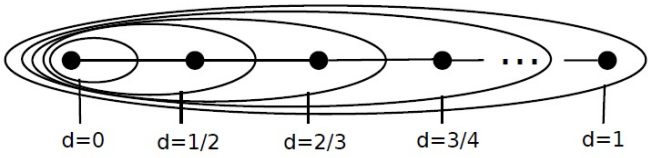
Treba podotknúť, že opakované kontrolovanie spojitosti spomaľuje beh algoritmu a preto ho môžeme v prípade potreby odstrániť a rozpojené komunity „zahadzovať".

Ako RaReIS, tak aj CIS vyprodukuje veľké množstvo veľmi sa prekrývajúcich komunít. Tento problém musíme odstraňovať vhodným post-processingom nájdených komunít. Veľkou výhodou CIS je však to, že optimalizačné procesy *seed* skupín sú na sebe nezávislé, preto môže byť algoritmus ľahko paralelizovaný.

Oproti CPM dovoľuje charakter CIS odhaliť komunity s rôznou štruktúrou a hustotou, ako už bolo naznačené na obr. 12, str. 25. Nejedná sa však o „dokonalú" metódu a to už len kvôli jej heuristickému prístupu, ktorý spôsobuje, že vygenerované komunity závisia na poradí spracúvaných vrcholov. Autori v článku [8] taktiež uvádzajú motivačný príklad, v ktorom ukazujú, že CIS (resp. lokálna optimalizácia) dokáže odhaliť aj kruhové komunity. To však nemusí byť vždy pravda, ako uvádzame ďalej v časti 4.2.3.

4.2.2 Upravená funkcia hustoty

Použitie základnej verzie funkcie hustoty (rovnica 7, 24) je problematické v prípade riedkych podgrafov. Na obr. 21 je uvedený príklad lokálnej optimalizácie zreťazených vrcholov. Hodnoty d vyjadrujú hustotu označenej časti, podľa základnej funkcie hustoty. Postupným zväčšovaním skupiny (na obr. z ľava do prava) sa každým pridaním ďalšieho vrcholu hustota skupiny vzhľadom na základnú verziu funckie hustoty zvyšuje. To má za následok, že celá „reťaz" vrcholov je detekovaná ako jedna komunita.



Obr. 21: Problém lokálnej optimalizácie zreťazených vrcholov.

Dalo by sa diskutovať o tom, či sa v takomto prípade majú jednotlivé dvojice/trojice vrcholov zaradiť do samostatných komunít, alebo či majú všetky patriť do jednej komunity. Autori kvôli tomuto problému upravili základnú verziu funkcie hustoty pridaním parametra, ktorým penalizujú pripojenia vrcholov, ktoré znižujú *pravdepodobnosť hrán* (z angl. *edge probability*) rozširovanej skupiny. Edge probability e_p skupiny S je vyjadrená ako

$$e_p = \frac{\sum\limits_{i,j \in S} e_{i,j}}{|S| \times (|S| - 1)}. \tag{15}$$

Upravená funkcia hustoty v CIS má potom tvar

$$d(S) = \frac{2 * w_{in}}{2 * w_{in} + w_{out}} + \lambda e_p, \tag{16}$$

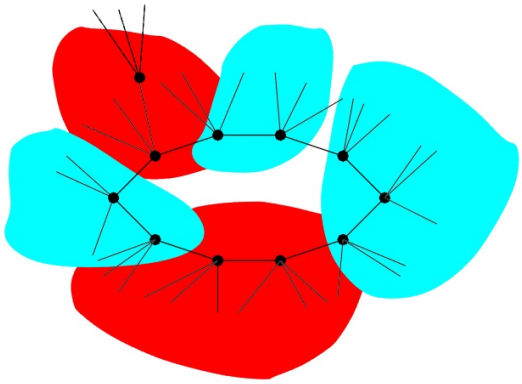
kde λ je parameter pomocou ktorého môžeme kontrolovať charakter výsledných komunít. V prípade $\lambda = 0$ dostaneme rovnaký výsledok, ako pri použití základnej verzie funkcie hustoty. So zvyšujúcou sa hodnotou λ zvyšujeme aj dôležitosť *internal edge probability* a dostávame naopak menšie komunity.

Namiesto počtu vnútorných w_{in} a externých hrán w_{out} sa v tejto funkcii hustoty používa vnútorný a vonkajší stupeň všetkých vrcholov, preto sa mení w_{in} na $2 * w_{in}$.

V nasledujúcej sekcii opíšeme problém, ktorý sme si všimli v popise článku [8].

4.2.3 Problém s motivačným príkladom

Autori v [8] na str. 3 uvádzajú, že pomocou lokálnej optimalizácie môžeme odhaliť veľké množstvo komunít s rôznymi hustotami. Ako motivačný príklad k tomuto tvrdeniu uvádzajú situáciu na obr. 22.



Obr. 22: Motivačný príklad lokálnej optimalizácie.

Na tomto obr. je každý vrchol patriaci do kruhovej komunity (ďalej len kruh) spojený s δ ďalšími vrcholmi mimo kruhu. Autori ukazujú, že vrcholy môžu patriť do kruhu a zároveň aj do iných externých komunít. Tvrdia, že aj keď je kruh riedky, tak je lokálne optimálny, čo dokazujú ďalej pomocou výpočtu hustoty po odstránení (resp. pridaní) vrcholu z kruhu. Ako funkcia hustoty bola pri tomto príklade použitá funkcia v základnom tvare (rovnica 7, str. 24).

Prvým problémom tohto príkladu je vzorec na výpočet zníženia hustoty kruhu po odstránení vrcholu u, ktorý autori uvádzajú v tvare

$$d(S-u) = \frac{1}{\delta + 1 + \frac{\delta}{|S|-2}}. \tag{17}$$

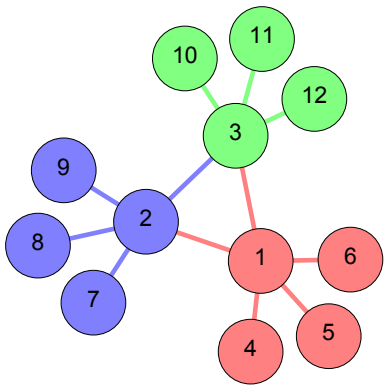
Nemyslia pri tom na to, že po odobraní vrcholu z kruhu sa tento vrchol neodstraňuje zo siete úplne a preto je treba jeho vnútorné hrany po odstránení započítavať ako vonkajšie. Správna verzia vzorca je

$$d(S-u) = \frac{1}{\delta + 1 + \frac{\delta + u_{in}}{|S|-2}}, \tag{18}$$

kde u_{in} je počet hrán vrcholu u patriacich do S. Hustota kruhu sa však naozaj po odobraní vrcholu zmenší.

Druhý problém, tohto príkladu nastáva pri pokuse navýšenia hustoty kruhu pridaním vrcholu. Autori správne uvádzajú, že pri pokuse pripojenia vrcholu, ktorý je mimo kruhu, sa hustota zmenší. Čo však môže byť zavádzajúce, je to, že pokles hustoty nastane len v prípade, ak má pripájaný vrchol z (z je ku kruhu pripojený len 1 hranou) viac hrán mimo kruhu, ako je *priemerný počet vonkajších hrán + 1* pre vrcholy v kruhu. Jedná sa o modelovú situáciu, ktorá môže zvádzať k myšlienke, že lokálnou optimalizáciou odhalíme riedke komunity vždy. To však nie je pravda.

Napr. pre graf na obr. 23, by sme očakávali, že vrcholy $1, 2, 3$ budú patriť okrem nájdených komunít na obr. aj do jednej spoločnej komunity. CIS však tieto vrcholy do spoločnej komunity nezaradil.



Obr. 23: Ukážka problému pri detekovaní riedkej komunity s CIS.

Netvrdíme, že motivačný príklad je nesprávny. V daných podmienkach sú uvedené tvrdenia správne (s výnimkou rovnice 17). Chceli sme len ukázať, že v tomto prípade, by bolo dobré poukázať na to, že lokálna optimalizácia nemusí riedke komunity rozpoznať vždy.

5 Implementácia

V tejto sekcii sa budeme venovať implementačným detailom vytvorenej aplikácie. Najskôr uvedieme všeobecný popis funkcionality a grafického používateľského rozhrania. Ďalej sa budeme venovať popisu implementácie metód CPM, CIS a dvom metódam, ktoré sme odvodili od CIS.

5.1 Všeobecný popis

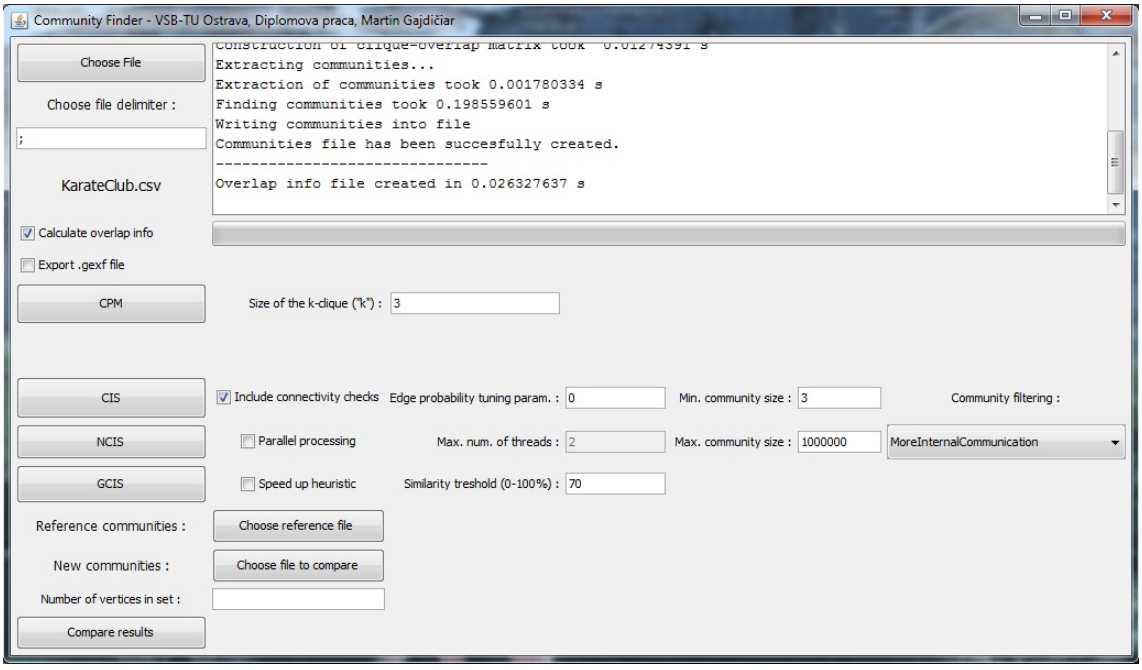
Primárnym cieľom aplikácie je detekcia komunít v komplexných sieťach, ktorá prebieha pomocou jednej z nasledujúcich implementovaných metód :

- CPM - verzia CPM, ako bola popísaná v 4.1, str. 34
- CIS - základná verzia CIS, ako bola popísaná v 4.2, str. 38
- NCIS - upravená verzia CIS, ktorá v každom skene vyhodnocuje len najbližšie okolie rozširovanej skupiny
- GCIS - greedy metóda založená na CIS doplnenom o iné poznatky získané z popisu ďalších metód (napr. 5.4.2, str. 55)

Okrem sekvenčných verzií týchto metód obsahuje aplikácia aj paralelnú implementáciu *CIS*, *NCIS* a *GCIS*.

Aplikácia umožňuje tiež :

- porovnanie výsledkov detekcie na základe rôznych metrík (viď 6.1, str. 59)
- export štatistík detekcie, medzi ktoré patria :
  - počet vrcholov a počet hrán skúmaného grafu
  - percentuálne pokrytie grafu komunitami
  - priemernú veľkosť, veľkosť najmenšej a veľkosť najväčšej komunity
  - počet komunít
  - počet vrcholov patriacich do nejakej komunity
  - počet prekrývajúcich sa párov komunít
  - počet vrcholov patriacich do viac ako jednej komunity
- export súboru, ktorý obsahuje dvojice prekrývajúcich sa komunít spolu s výpisom ich spoločných vrcholov
- export *.gexf* súboru [48], ktorý je určený na vizualizáciu siete a nájdených komunít v programe Gephi [49].



Obr. 24: Ukážka grafického používateľského rozhrania aplikácie.

Dáta sú pre vizualizáciu v Gephi spracované tak, aby mala každá komunita inú farbu a prekrývajúce sa vrcholy mali väčšiu veľkosť ako normálne.

Ukážka jednoduchého grafického používateľského rozhrania (GUI) aplikácie je na obr. 24 V hornej pravej časti GUI sa nachádza informačný panel zobrazujúci dôležité informácie a priebeh algoritmov, ďalej je tu umiestnený progress bar indikujúci, že aplikácia pracuje. V ľavej hornej časti sa nachádzajú všeobecné nastavenia spoločné pre všetky metódy :

- *Choose file* - výber súboru s grafom (špecifický formát je uvedený v 5.2.1), ktorý má byť spracovaný
- *Choose file delimiter* - oddeľovač použitý v súbore s grafom
- názov spracovávaného súboru
- *Calculate overlap info* - checkbox určujúci, či sa majú vygenerovať dodatočné informácie
- *Export .gexf file* - checkbox určujúci, či sa má vygenerovať .gexf súbor určený pre vizualizáciu v Gephi

Hneď pod vrchnou časťou sa nachádzajú ovládacie prvky spojené s metódou CPM :

- *CPM* - tlačidlo slúžiace na spustenie CPM

- *Size of the k-clique („k“)* - veľkosť k-kliky

Nasleduje časť obsahujúca potrebné nastavenia a ovládanie CIS a odvodených metód :

- *CIS, NCIS, GCIS* - tlačidlá na spustenie jednotlivých metód
- *Include connectivity checks* - checkbox určujúci, či sa majú po skenoch vykonávať kontroly konektivity
- *Parallel processing* - checkbox určujúci, či sa má spustiť paralelná verzia metódy
- *Speed up heuristic* - checkbox určujúci, či sa má pri vykonávaní metódy použiť heuristika na zrýchlenie výpočtu
- *Edge probability tuning param* - hodnota ladiaceho parametra λ (viď 16, str. 41)
- *Max. num. of threads* - maximálny počet vlákien, ktoré majú byť použité pri paralelnom výpočte
- *Similarity threshold (0-100%)* - určuje hranicu percentuálnej podobnosti, nad ktorou sa majú 2 komunity zlúčiť do jednej (viď 5.4.0.1, str. 53)
- *Min.* a *max. community size* - hodnoty obmedzujúce veľkosť hľadaných komunít v danom rozsahu
- *Community filtering* - drop-down list s hodnotami *None, More internal communication, More external communication*, ktoré určujú, či sa majú zisťovať len komunity s väčšou vnútornou komunikáciou, vonkajšou komunikáciou, alebo všetky

5.2 Implementačné základy

Aplikácia je vytvorená v programovacom jazyku Java. Na správu závislostí je použitý Maven.

Keďže sme sa snažili o to, aby bola implementácia čo najefektívnejšia, tak bol minimalizovaný počet objektov a ich vlastností. Okrem toho sme namiesto základných kolekcií, ktoré poskytuje JDK používali kolekcie z knižnice *Trove* [47]. Problémom v prípade použití klasických Java kolekcií je, že sú *Object-based*. To znamená, že pre primitívne typy sa používa tzv. *wrapper class*, napr. pre *int* je to *java.lang.Integer*. Pre väčšinu aplikácií však chceme uchovávať primitívne typy priamo, vďaka čomu spotrebujeme menej pamäťového miesta a značne zvýšime výkonnosť. Knižnica Trove presne toto umožňuje. Obsahuje „lightweight“ implementáciu kolekcií s veľmi podobným API ako má *java.utils.Collection*.

Triedny diagram zobrazujúci najdôležitejšie triedy je v prílohe A. Popis uvedených tried je nasledujúci :

- *GraphExporter* - obsahuje metódy slúžiace na export grafu s farebne odlíšenými komunitami do *.gexf* súboru

- *InfoCalculator* - vypočítava dodatočné informácie o nájdených komunitách a určuje prekrývajúce sa komunity s ich vrcholmi
- *TextAreaUpdaterUtil* - aktualizuje obsah informačného panela
- *Metrics* - predstavuje jednotlivé metriky
- *FileWriterUtil* - obsahuje metódy na vytváranie jednotlivých súborov
- *CISBase* - abstraktná trieda, ktorá obsahuje metódy spoločné pre CIS, NCIS a GCIS. Ďalej obsahuje abstraktné metódy *normalProcessing* a *parallelProcessing*, ktoré majú byť implementované, pričom predstavujú sekvenčný a paralelný chod daných typov algoritmov
- *NCIS, GCIS, CIS* - obsahujú implementáciu jednotlivých algoritmov
- *NCISRunnableTask, GCISRunnableTask, CISRunnableTask* - obsahujú implementáciu spúšťaných paralelných úloh
- *Graph* - rozhranie s metódami, ktoré slúžia na prácu s grafom
- *AdjacencyListGraph* - konkrétna implementácia rozhrania *Graph* založená na *adjacency liste* (viď 2.1.6.3, str. 14)
- *CPM* - obsahuje implementáciu metódy CPM
- *MainProgram* - hlavné okno programu
- *MetricsCalculator* - obsahuje metódy na výpočet jednotlivých metrík (viď 6.1, str. 59)
- *Parser* - rozhranie obsahujúce metódy na spracovanie súboru s grafom a s komunitami, na základe ktorých vytvorí konkrétny graf a *HashMap* komunít, v ktorej je kľúč číslo riadku na ktorom sa komunita v súbore nachádza a hodnota je zoznam vrcholov komunity
- *NormalParser* - implementuje rozhranie *Parser*, pričom reprezentuje načítané komunity ako *String*

5.2.1 Práca s grafmi

V programe je vrchol reprezentovaný len svojím ID v podobe *int* premennej. ID vrcholov sú vygenerované pri spracovaní súboru obsahujúceho graf. Aplikácia pracuje so súbormi, v ktorých sú grafy definované pomocou dvojíc vrcholov (čo znamená, že medzi nimi existuje hrana) na jednotlivých riadkoch. Všetky dvojice vrcholov musia byť oddelené pomocou rovnakého oddeľovača (napr. medzera, čiarka a pod.). Pri spracovaní súboru (trieda *NormalParser*, metóda *parseFile(String pathToFile, String delimiter) : Graph*) vzniká mapovanie vygenerovaného ID vrcholu na reálnu hodnotu vrcholu, ktoré sa ukladá do statickej HashMap.

Počas spracovania sa taktiež vytvára samotný graf (inštancia triedy *AdjacencyListGraph*) s ktorým budeme ďalej pracovať. Keďže predpokladáme, že budú pomocou aplikácie analyzované veľké siete, tak sme sa grafy rozhodli reprezentovať pomocou zoznamu susedov (viď 2.1.6.3, str. 14). Ten je implementovaný ako HashMap, v ktorej kľúče sú ID vrcholov grafu a hodnoty sú Listy ich susedných vrcholov. Na urýchlenie vyhľadávania konkrétneho suseda sme jednotlivé Listy zotriedili podľa ID vrcholov, vďaka čomu môže byť použité binárne vyhľadávanie.

Zdrojový kód spracovania súboru a vytvorenia grafu sa nachádza v prílohe D, výpis č. 7.

Po ukončení akejkoľvek metódy sa vytvára zložka s textovým súborom, ktorý obsahuje použité nastavenia a časy detekcie komunít a iných úkonov (ako napr. čas exportu .gexf súboru). Ďalej zložka obsahuje .csv súbor s nájdenými komunitami. V súbore s komunitami sa každý riadok začína s identifikátorom komunity a jej veľkosťou v tvare *ID_veľkosť* a ďalej nasledujú na riadku vrcholy komunity. Napr.: 1_4, 1, 8, 7, 3.

Používateľ má taktiež možnosť vygenerovať dodatočné informácie o nájdených komunitách, ako bolo uvedené v 5.1.

Vizualizáciu výsledkov sme vytvárali pomocou programu Gephi [49]. Preto má používateľ aplikácie možnosť vygenerovať po vyhľadaní komunít aj .gexf [48] súbor obsahujúci celý graf s farebne odlíšenými komunitami, pričom vrcholy patriace do viacerých komunít majú väčšiu veľkosť.

Pôvodne sme chceli prekrývajúcim sa vrcholom nastaviť iný tvar, alebo zmiešanú farbu, ale Gephi tieto možnosti momentálne (verzia 0.8.2 beta) neumožňuje. Nastavenie úplne inej farby ako majú komunity do ktorých vrchol patrí, by mohlo v prípade veľkého prekrytia spôsobiť to, že prekrývajúce vrcholy budú kvôli jednotnej farbe vypadať, ako keby tvorili samostatnú komunitu. Preto sme sa rozhodli len pre zväčšenie veľkosti prekrývajúcich sa vrcholov, aj keď toto riešenie nie je ideálne a v budúcnosti by malo byť vylepšené.

Pri generovaní .gexf súboru používame knižnicu *gexf4j* [50]. Zdrojový kód exportu .gexf súboru sa nachádza v prílohe D, vo výpise č. 8. Farby pre komunity sú vygenerované jednoducho, pomocou zmeny odtieňa (Hue) v HSB farebnom priestore. Zdrojový kód na vygenerovanie farieb je ukázaný vo výpise č. 1.

```
private static List<Color> generateColors(int count) {
    List<Color> colors = new ArrayList<Color>();
    float step = 1f / count;
    for (float i = 0; i < 1; i += step) {
        colors.add(Color.getHSBColor(i, 0.5f, 1f));
    }
    return colors;
}
```

Výpis 1: Vygenerovanie daného počtu rôznych farieb

5.3 Implementácia CPM

CPM sme implementovali tak, že pri každom spustení analyzujeme graf, resp. vyhľadávame kliky pre konkrétnu hodnotu *k*, vďaka čomu môžeme pred samotným vyhľadaním klík vykonať redukciu vrcholov a zrýchliť tak algoritmus, čo je veľmi výhodné najmä pri analýze veľkých grafov. Napr. pri vyhľadávaní komunít pre hodnotu $k = 10$ nemá zmysel uvažovať vrcholy, ktoré majú menší stupeň ako 9, všeobecne potom $k - 1$. To z toho dôvodu, že ak chceme aby vrchol patril do k-kliky tak musí mať aspoň $k - 1$ susedov.

Na vyhľadanie maximálnych klík grafu sme použili algoritmus Bron-Kerbosch popísaný v sekcii 4.1.2, str. 37. Zdrojový kód implementácie algoritmu je ukázaný v prílohe D vo výpise č. 9.

Počas implementácie CPM sme narazili na problém efektívnej reprezentácie *clique-clique overlap matice*. Rýchlo sme zistili, že jednoduché 2D pole nie je kvôli veľkým pamäťovým nárokom u väčších grafov vhodné. Prvým krokom k optimalizácii bolo využitie faktu, že matica je *symetrická*. Reprezentovali sme ju teda pomocou skoseného 2D poľa (*jagged array*), v ktorom majú riadky rôzny počet stĺpcov (ukladáme len hodnoty nad diagonálou). Hodnoty pod diagonálou potom dopočítame podľa hodnôt nad diagonálou. Vyplnená matica potom môže vyzerať napr. takto :

$$\begin{pmatrix} 1 & 0 & 0 & 1 \\ x & 1 & 0 & 0 \\ x & x & 1 & 1 \\ x & x & x & 1 \end{pmatrix},$$

kde prvky x znamenajú prázdne miesto. Dosiahli sme zlepšenie výkonnosti, ale pre väčšie grafy (cez 80 tisíc vrcholov) to stále nebolo dostačujúce.

Okrem toho, že je clique-clique overlap matica symetrická, tak je aj *riedka*. Ďalším optimalizačným krokom bolo preto vytvorenie riedkej symetrickej matice reprezentovanej pomocou HashMap, kde kľúč tvorí jedinečný identifikátor prvku riadka a stĺpca (*riadok+stĺpec∗(veľkosť_matice+1)*), v ktorom je hodnota veľkosti prekrytia klík dostatočná (*diagonálny prvok* $>$ *k* , *ne-diagonálny prvok* $>$ *(k-1)*, viď teória 4.1.1, str. 35). Nielenže teda neukladáme nulové prvky, ale aj prvky, ktoré nespĺňajú podmienku dostatočného prekrytia. Takáto matica potom vyzerá nasledovne :

$$\begin{pmatrix} 1 & x & x & 1 \\ x & 1 & x & x \\ x & x & 1 & 1 \\ x & x & x & 1 \end{pmatrix},$$

Tým, že pri generovaní kľúča HashMapy prenásobujeme stĺpec číslom väčším ako je rozmer matice sa vyhneme duplicitám. Napr. v prípade matice rádu 4 by pri prenásobení stĺpca číslom 3 vznikol konflikt pre kľúč prvku $[r0][s1]$ a $[r3][s0]$.

Použitím takto implementovanej matice sme dosiahly niekoľkonásobné (až 6x) zvýšenie efektivity. Zdrojový kód prípravnej fázy matice je uvedený vo výpise č. 2.

```
TIntByteHashMap cliqueOverlapMatrix = new TIntByteHashMap(maximalCliques.size());
int row = 0;
int overlapSize = 0;
for( TIntList clique : maximalCliques){
    for (int col = 0; col < maxCliqueSize−row; col++){
        // diagonalny element, pretoze dalej ziskavame kliku na pozicii (col+row). Kazdy dalsi
            riadok sa posuvame o stlpec doprava.
        if (col == 0){
            overlapSize = clique.size() ;
            overlapSize −= k;
        }
        else {
            overlapSize = retainAllCustom(clique, maximalCliques.get(col+row)).size();
            overlapSize −= (k−1);
        }
        if (overlapSize >= 0)
            cliqueOverlapMatrix.put(row + col ∗ hashMultiplier, OVERLAP_IS_SUFFICIENT);
    }
    row++;
}
```

Výpis 2: Príprava clique-clique overlap matice

Pred vytvorením matice boli všetky nájdené kliky z ktorých sa matica vytvárala uložené do ArrayListu $lastMaximalCliques$. Po vytvorení matice už len stačí zlúčiť kliky riadkov podľa postupu popísaného v 4.1.1, str. 35.

Prechádzame postupne jednotlivé riadky matice a zlučujeme kliky daného riadka do dočasnej „riadkovej komunity“ K_r, ktorú pridávame do zoznamu riadkových komunít $rowCommunities$. Na každom riadku (okrem prvého) pracujeme aj so symetrickými hodnotami z predchádzajúcich riadkov. Pri zlučovaní klík jednotlivých riadkov pracujeme len s indexami klík (pozícia kliky v ArrayListe $lastMaximalCliques$) a nie so samotnými klikami. Po spracovaní každého riadku kontrolujeme, či sa prípadná novo vytvorená riadková komunita K_r nedá zlúčiť s už existujúcimi riadkovými komunitami v $rowCommunities$. Ilustračný príklad vid' d'alej.

Příklad 5.1
ArrayList $lastMaximalCliques$ obsahuje kliky A,B,C,D,E,F. Na prvom riadku matice sme zlúčili do riadkovej komunity K_1 indexy klík A,B,C, $rowCommunities$ teda obsahuje $K_1 = \{0, 1, 2\}$. Ďalej sme na druhom riadku zlúčili do riadkovej komunity K_2 kliky C,D, $K_2 = \{2, 3\}$ a kontrolujeme, či sa K_2 nedá zlúčiť s niektorou riadkovou komunitou z $rowCommunities$. V tomto prípade kontrolujeme, či $K_1 \cap K_2 \neq \emptyset$, čo platí, pretože majú spoločný index 2 (kliku C). Preto ich zlúčime do jednej riadkovej komunity $K_{new} = K_1 \cup K_2$, pridáme ju do $rowCommunities$ a odstránime z $rowCommunities$ komunitu K_1. ■

Zdrojový kód implementácie získania komunít z matice je uvedený v prílohe D, výpis č. 10.

Správnosť implementácie sme testovali na menších testovacích grafoch a pre väčšie grafy sme naše výsledky porovnávali s výsledkami získanými pomocou známeho nástroja CFinder [51], pričom sa zhodovali.

5.4 Implementácia CIS

Najskôr sme implementovali základnú verziu CIS, tak ako bola popísaná v sekcii 4.2.1. Každý vrchol je zaradený do svojej vlastnej *seed* group, pričom sú spracovávané podľa stupňa od najmenšieho po najväčší. V každom skene sa na odobranie alebo pridanie k seed skupine zvažuje každý vrchol grafu. Preto je výpočet časovo veľmi náročný a neefektívny. Ak má graf n vrcholov a pre každý sa vykoná k skenov (zovšeobecnené), v ktorých sa vyhodnocujú všetky vrcholy grafu práve raz, tak počet operácií sa rovná $n(nk)$, čo je zložitosť $O(n^2k)$. Zložitosť sa však určuje len veľmi ťažko, pretože záleží na hustote skúmanej siete a štruktúre jej komunít.

Ukážka zdrojového kódu znázorňujúceho priebeh jedného skenu je vo výpise č. 3.

```
for (int it = 0; it < sortedVertLength; it++) {
    comparedVertex = sortedVertices[it];
    if (!isVertexConnectedToCommunity(comparedVertex, comm))
        continue;
    if (comm.size() > 2 && comm.remove(comparedVertex)) {
        newDensity = calculateDensity(comm, tuningParam);
        if (!( newDensity > oldDensity)) {
            comm.add(comparedVertex);
            comm.sort();
        } else {
            oldDensity = newDensity;
            improved = true;
        }
    }
    else if (comm.binarySearch(comparedVertex) < 0) {
        comm.add(comparedVertex);
        comm.sort();
        newDensity = calculateDensity(comm, tuningParam);
        if (!( newDensity > oldDensity))
            comm.remove(comparedVertex);
        else {
            oldDensity = newDensity;
            improved = true; // dalej nasleduje len ukoncenie podmienok a cyklu
```

Výpis 3: Priebeh skenu metódy CIS

Hustota sa vypočítava podľa vzťahu 16, str. 41. Skeny prebiehajú pokiaľ došlo v predchádzajúcom skene k zlepšeniu hustoty komunity. Po ukončení každého skenu sa kontroluje konektivita výslednej skupiny (metóda *connectivityCheck(TIntList comm, double tuningParam)*). Ak sa skupina skladá z viacerých komponentov (je rozpojená), tak sa ďalej pracuje komponentom, ktorý má najväčšiu hustotu. Jednotlivé komponenty sú vyhľadané pomocou prehľadávania do šírky (BFS, viď 2.1.5, str. 2.1.5) v metódach *findConnectedComponents(TIntList comm)* a *bfsAdd(TIntList comm, int vertex, TIntList comp)*, kde *comm*

je skúmaná skupina, *vertex* je vrchol od ktorého začíname BFS a *comp* je nájdená komponenta. Ukážka zdrojového kódu týcho metód je vo výpise č. 4.

```
public List<TIntList> findConnectedComponents(TIntList comm) {
    List<TIntList> connectedComps = new ArrayList<TIntList>();
    TIntList comp = new TIntArrayList();
    for (int commIt = 0; commIt < comm.size(); commIt++) {
        int vertex = comm.get(commIt);
        if (isVertexContainedInComponents(vertex, connectedComps))
            continue;
        comp = new TIntArrayList();
        bfsAdd(comm, vertex, comp);
        connectedComps.add(comp);
        // ak najdena komponenta pokryva celu komunitu tak nepokracujeme dalej v overovani
        if (comp.size() == comm.size())
            break;
    }
    return connectedComps;
}

public void bfsAdd(TIntList comm, int vertex, TIntList comp) {
    Queue<Integer> q = new LinkedList<Integer>();
    q.add(vertex);
    while (!q.isEmpty()) {
        int currVertex = q.poll();
        TIntList vertexNeighbours = tgraph.getVertexNeighbors(currVertex);
        for (int neighIt = 0; neighIt < vertexNeighbours.size(); neighIt++) {
            int vertexNeighb = vertexNeighbours.get(neighIt);
            if (comm.binarySearch(vertexNeighb) >= 0 && comp.binarySearch(vertexNeighb) < 0 &&
                !q.contains(vertexNeighb)) {
                q.add(vertexNeighb);
            }
        }
        comp.add(currVertex);
        comp.sort();
    }
}
```

Výpis 4: Vyhľadanie komponent skupiny

Po celkovom ukončení skenov nad jednou seed skupinou prebieha vyhodnotenie finálnej podoby rozširovanej skupiny (metóda *afterScanProcess(...)*). V tomto vyhodnotení sa kontroluje, či skupina spĺňa požadované parametre (min./max. veľkosť, typ komunikácie, konektivita). V prípade, že parametre nespĺňa, tak ju neuvažujeme.

Jediná výnimka je, ak má skupina väčšiu veľkosť ako je max. stanovená. Vtedy jej veľkosť zredukujeme tak, že postupne odstraňujeme vrcholy, ktoré spôsobia najmenšie zhoršenie hustoty, pokiaľ sa nedosiahne požadovaná veľkosť. Ak výsledná skupina vyhovuje všetkým požiadavkám, tak je zotriedená a zaradená do HashSetu finálnych komunít.

Tým, že sa skupina pred pridaním do HashSetu finálnych komunít zotriedi sa HashSet postará o filtráciu duplicitných komunít. CIS však detekuje veľké množstvo prekrývajúcich sa komunít a preto je potrebné finálne komunity ďalej spracovať.

5.4.0.1 Post processing komunít V post processingu sa najskôr odstraňujú komunity, ktoré pokrývajú celý graf. Vďaka zotrieďovaniu a HashSetu taká môže byť v kolekcii len 1 a odstráni sa len pokiaľ sa našla aj nejaká iná komunita okrem nej. Pre malé grafy totiž často platí, že sú prepojené tak veľa, že obsahujú len jednu veľkú komunitu.

Ďalej sa odstraňujú komunity, ktoré sú podmnožinou inej komunity (naznačuje to určitú formu hierarchie, ktorá by sa dala využiť na stanovenie hierarchie v jednotlivých komunitách siete). Nakoniec má používateľ možnosť zadať percentuálnu hranicu prekrytia, pri ktorej sa dve prekrývajúce sa komunity zlúčia do jednej.

Podobnosť medzi dvoma prekrývajúcimi sa komunitami A a B sme najskôr určovali pomocou *Jaccardovho indexu* určeného ako :

$$J(A,B) = \frac{A \cap B}{A \cup B}. \tag{19}$$

Jaccardov index nadobúda hodnoty od 0 po 1, s tým, že čím bližšie je hodnota k 1, tým viac sa množiny podobajú.

Problém bol však v tom, že ak sme mali komunitu A s vrcholmi 1,2,3, označme $A = \{1,2,3\}$ a $B = \{2,3,4,5,6,7,8,9,10\}$, tak $J(A,B) = 0.2$, čo je malá hodnota na to, že A zdieľa s B 2 z 3 svojich vrcholov. Veľkosť jednej z komunít ovplyvňovala celkovú hodnotu indexu. Preto sme sa rozhodli určovať percentuálnu mieru zdieľaných vrcholov pre každú z dvojice komunít a podobnosť určiť ako väčšiu z týchto dvoch hodnôt. Podobnosť S je potom určená ako :

$$S(A,B) = max\left(\frac{A \cap B}{|A|}, \frac{A \cap B}{|B|}\right). \tag{20}$$

V tomto prípade sa už nedá hovoriť o podobnosti ako takej, ale skôr o miere zdieľania vrcholov. Tento postup sa podobá určovaniu vzdialenosti medzi komunitami použitému v metóde GCE poppísanej v sekcii 3.3.3.1, str. 31.

Zlučovanie prekrývajúcich sa komunít, alebo priraďovanie prekrývajúcich sa vrcholov len do jednej komunity však nie je jednoduchá úloha. V rôznych sieťach sa komunity môžu prekrývať viac, alebo menej, pričom samotná veľkosť prekryvu nemusí vypovedať o jeho dôležitosti. Preto nie je možné stanoviť univerzálnu mieru podobnosti vhodnú pre všetky siete.

Ukážka zdrojového kódu post processingu je v prílohe D, výpis č. 11.

Kvôli neefektivite metódy CIS pre veľké grafy sme ju z časti upravili, čím vznikli 2 ďalšie odvodené metódy NCIS a GCIS uvedené ďalej.

5.4.1 NCIS

NCIS (Neighbors CIS) sa líši od CIS dvoma úpravami.

Prvá úprava CIS bola tá, že sme pri skenoch prechádzali namiesto všetkých vrcholov grafu len susedné vrcholy aktuálne rozširovanej komunity (získané metódou *getCommunityVerticesAndNeigbors(TIntList comm)*). Keďže CIS je v podstate heuristika u ktorej výsledky záležia na poradí spracovania vrcholov, tak sme touto zmenou dostali s NCIS

trocha iné výsledky ako u CIS (porovnanie bude uvedené v časti s experimentami 6, str. 58). Rapídne sa však zrýchlil chod algoritmu.[3]

Kvôli tomu, že CIS zaraďuje každý vrchol grafu do samostatnej seed skupiny, sú tieto skupiny optimalizované často k už nájdenej komunite.

Příklad 5.2
Uvažujme trojuholníkový graf (3 vrcholy, každý rádu 2) o vrcholoch $V = \{1, 2, 3\}$. Optimalizovaním z vrcholu 1 dostaneme v dvoch krokoch pridaním 2 a 3 komunitu $K = \{1, 2, 3\}$, optimalizovaním z 2 a 3 dostaneme rovnakú komunitu K v ďalších, zbytočných 4 krokoch. ■

Keďže je veľká pravdepodobnosť na to, že z vrcholov, ktoré sme už zaradili do nejakej komunity dostaneme optimalizovaním rovnakú komunitu, tak takéto vrcholy ďalej nepovažujeme za seed skupinu.

Samozrejme tým, že sa komunity prekrývajú nemusíme z vrcholu zaradeného do nejakej komunity dostať tú istú komunitu. Takéto vrcholy ale tiež nemusíme rozširovať viď príklad 5.3 ďalej.

Příklad 5.3
Uvažujme komunity $K_1 = \{A, B, C\}$ a $K_2 = \{B, C, D\}$. Tie sa prekrývajú na vrchole B. Rozširovaním z A sme našli komunitu K_1. Nemusíme sa však báť, že neodhalíme K_2, alebo prekrytie, tým, že preskočíme rozširovanie z už zaradeného vrcholu B. Oboje odhalíme rozširovaním z vrcholu C alebo D. ■

Druhá úprava CIS bola teda zavedenie uvedenej „urýchľovacej“ heuristiky. Podobne riešili tento problém aj autori v LFM 3.3.2, str. 28. Tí však vyberali vrcholy z grafu náhodne, zatiaľ čo my postupujeme podľa stupňa vrcholov od najmenšieho po najväčší, rovnako ako v CIS.

Možnosť použiť túto heuristiku sme nakoniec zaviedli do všetkých variánt CIS (v GUI checkbox *Speed up heuristic*).

Zdrojový kód úprav v NCIS oproti CIS je ukázaný vo výpise č. 5.

```
for (int i = 0; i < sortedVertLength; i++) {
    vertex = sortedVertices[i];
    // vrchol spracovavame len ak uz nebol zaradeny do nejakej komunity
    if (speedUp && processedVertices.contains(vertex))
        continue;
...
// v skene chceme vyhodnocovat len vrcholy rozsirovanej skupiny a ich susedov
TIntList communityVerticesAndNeighb = getCommunityVerticesAndNeigbors(comm);
// samotny cyklus skenu
for (int it = 0; it < communityVerticesAndNeighb.size(); it++) {
    comparedVertex = communityVerticesAndNeighb.get(it);
...
// metoda na ziskanie vrcholov a susedov rozsirovanej skupiny
public TIntList getCommunityVerticesAndNeigbors(TIntList community) {
```

[3] Treba poznamenať, že toto zlepšenie sme vymysleli samy, pričom sme po vypracovaní práce zistili, že rovnaké vylepšenie bolo vykonané už v [11].

```
TIntList sortedVerticesToReturn = new TIntArrayList(community.size() * 3);
for (int i = 0; i < community.size(); i++) {
    int vertex = community.get(i);
    if (!sortedVerticesToReturn.contains(vertex))
        sortedVerticesToReturn.add(vertex);
    TIntList vertexNeighb = tgraph.getVertexNeighbors(vertex);
    for (int j = 0; j < vertexNeighb.size(); j++) {
        if (!sortedVerticesToReturn.contains(vertexNeighb.get(j)))
            sortedVerticesToReturn.add(vertexNeighb.get(j));
    }
// nasleduje usporiadanie a vratenie vrcholov sortedVerticesToReturn ...
```

Výpis 5: Zmeny v NCIS oproti CIS

5.4.2 GCIS

V metóde GCIS (Greedy CIS) sme spojili zmeny vytvorené v NCIS s ďalšími poznatkami získanými z iných metód spomínaných v teoretickej časti. Inšpirovali sme sa hlavne myšlienkou greedy optimalizácie rozširovanej skupiny z LFM 3.3.2, str. 28.

K rozširovanej seed skupine nepridávame automaticky každý vrchol, ktorý zlepší jej hustotu, ale vyberáme vždy taký, ktorý v aktuálne prebiehajúcom skene zlepší hustotu najviac. Takýmto greedy prístupom čiastočne odstránime nedeterminističnosť CIS.

Príkladom môže byť obr. 23 zo str. 43. V prípade CIS a NCIS postupujeme pri vyhodnocovaní seed skupín a vrcholov v skene podľa stupňa vrcholu od najmenšieho po najväčší. Takýmto spôsobom nájdeme 3 komunity, ktoré sú na obr. 23 farebne vyznačené (prípadne 2, záleží na tom, v akom poradí sa vyhodnotia vrcholy s rovnakým stupňom). Ak by sme však postupovali opačne, od najväčšieho stupňa po najmenší, tak by sme našli len 1 komunitu, ktorá by pokrývala celý graf.

Pri použití greedy prístupu je jedno v akom poradí vrcholy v skene prechádzame, pretože na zlepšenie hustoty vyberáme vždy vrchol, ktorý svojim pridaním, alebo odobraním zlepší hustotu skupiny najviac. Takýmto spôsobom by sme mali dostať najoptimálnejšie komunity.

Priebeh skenu sa u GCIS skladá z dvoch hlavných krokov :

1. Nájdenie vrcholu zo susedov skupiny, ktorý najviac zvýši hustotu skupiny. Ak takýto vrchol existuje tak ho pridáme ku skupine.

2. Odstránenie všetkých vrcholov skupiny, ktorých odstránením zlepšíme hustotu skupiny.

Ukážka zdrojového kódu priebehu skenu v GCIS je v prílohe D vo výpise č. 12.

Na prvý pohľad je jasné, že greedy prístup má aj svoju nevýhodu, ktorou je zpomalenie vyhľadávania oproti NCIS. V NCIS môžeme v jednom skene pridať neobmedzený počet vrcholov, zatiaľ čo v GCIS len 1.

Aby sme čo najefektívnejšie využili zrýchľovaciu heuristiku a vo výsledku dostali menší počet prekrývajúcich sa komunít, tak seed skupiny (jednotlivé vrcholy grafu) vyhodnocujeme od najväčšej po najmenšiu (podľa stupňa). Takto rýchlejšie vytvoríme po-

krytie grafu. Ideálne by bolo seed skupiny vopred vygenerovať tak, aby sa čo najmenej prekrývali a zároveň pokrývali celý graf. Pre porovnanie s ostatnými metódami sme však použili rovnakú voľbu seed skupín ako v CIS, kde každý vrchol patrí do samostatnej seed skupiny. Prípravu a testovanie lepších seed skupín ponechávame ako rozšírenie tejto práce.

5.4.3 Paralelné verzie

Vďaka tomu, že optimalizovanie jednotlivých seed skupín prebieha nezávisle na sebe, tak sú všetky metódy odvodené od CIS, teda CIS, NCIS a GCIS ľahko paralelizovateľné.

Používateľ má v GUI možnosť zadať maximálny počet vlákien použitých pri paralelizácii. Tá prebieha tak, že sa pre každú seed skupinu spustí nezávislá paralelná úloha (triedy *CISRunnableTask, GCISRunnableTask, NCISRunnableTask*), v ktorej prebieha optimalizácia danej skupiny na základe zvolenej metódy. Keďže vopred poznáme počet seed skupín, tak vieme koľko úloh sa musí dokončiť, aby bol výpočet kompletný. Vopred si teda incializujeme *CountDownLatch* na veľkosť počtu seed skupín, ktorý každá paralelná úloha po svojom dokončení zmenší o 1. Po znížení CountDownLatch na 0 vieme, že všetky paralelné úlohy sa skončili a môžeme prejsť ku post processingu nájdených komunít.

Skrátená ukážka zdrojového kódu paralelného spracovania je vo výpise č. 6.

```
...
ConcurrentHashMap<TIntList, Boolean> cs = new ConcurrentHashMap<TIntList, Boolean>();
final Collection<TIntList> communities = Collections.newSetFromMap(cs);
sortedVertices = unsortedVertices.toArray();
final int sortedVertLength = sortedVertices.length;
ExecutorService exec = Executors.newFixedThreadPool(numOfThreads);
final CountDownLatch latch = new CountDownLatch(sortedVertLength);
try {
    // kazdy uzol grafu patri do svojej vlastnej "seed group"
    for (int i = 0; i < sortedVertLength; i++) {
        Runnable worker = new CISRunnableTask(this, communities, sortedVertices,
                processedVertices, sortedVertices[i], tuningParam, includeConnectivityChecks,
                filteringMode, maxSize, minSize, latch, speedUp);
        exec.execute(worker);
    }
} finally {
    exec.shutdown();
}
try {
    // pockanie na dokoncenie jednotlivych taskov
    latch.await();
}
...

// v samotnom tasku sa po jeho dokonceni vola
latch.countDown();
```

Výpis 6: Ukážka paralelného spsracovania CIS

V prípade použitia urýchľovacej heuristiky však paralelizácia nie je vhodná kvôli tomu, že jednotlivé paralelné úlohy musia pracovať so spoločným zoznamom vrcholov, ktoré už boli zaradené do nejakej komunity (pričom ho používajú na zapisovanie a načítavanie veľmi často). Použitie heuristiky samotnej je rýchlejšie, ako jej použitie v spojení s paralelizáciou.

6 Experimenty

V tejto časti popíšeme vykonané experimenty nad rôznymi reálnymi sieťami.

Parametre testovacej zostavy :

- Procesor - Intel(R) Core(TM) i7-2670QM CPU @ 2.20GHz
- RAM - 8 GB
- Operačný systém - Windows 7 64bit

Skratky použité v tabuľkách s výsledkami experimentov :

- *po redukc.* - počet siete vrcholov po redukcii
- *komunity* - počet komunít
- *min., max., avg.* - všeobecne minimum, maximum, priemer
- *min. kom., max. kom., avg. kom.* - min.,max.,avg. veľkosť komunity
- *min. hust., max. hust., avg. hust.* - min.,max.,avg. hodnota hustoty komunity
- *pokrytie* - percentuálne pokrytie grafu komunitami
- *vrch v kom.* - počet vrcholov v komunitách
- *prekrýv. páry kom.* - počet prekrývajúcich sa párov komunít
- *prekrýv. vrch.* - počet prekrývajúcich sa vrcholov
- *base* - referenčná komunita

Ak nie je uvedené inak, tak pre CPM sa používa nastavenie parametra $k = 3$ (3-kliky) a pre CIS, NCIS a GCIS je min. veľkosť komunity = 1 a vyhľadávajú sa len komunity s väčšou vnútornou komunikáciou. Zavádzame nasledujúce značenia :

- S (speed) pred názvom metódy znamená, že bola použitá urýchľovacia heuristika
- B (best) znamená, že sa jedná o výsledky najlepšej konfigurácie

Ak v tabuľke výsledkov nie je uvedená B varianta nejakej metódy, znamená to, že najlepšie výsledky sme dosiahli so základnou verziou.

Pred uvedením samotných experimentov je nutné poznamenať, že validitu jednotlivých komunít overujeme len na základe ich štruktúry, teda v prípade CPM kombinácie k-klík a v prípade ostatných metód kombinácie vrcholov s najlepšou hodnotou funkcie hustoty. Predpokladáme, že dáta jednotlivých sietí, na ktoré bude naša aplikácia použitá, budú korektne predspracované a to v tom zmysle, že v grafe budú len relevantné vrcholy a hrany.

Pre experimenty sme zvolili niekoľko dobre známych menších sietí, no výkonnosť aplikácie sme otestovali aj na väčších sieťach. Pri vyhodnocovaní nájdených komunít

sme u sietí s neznámymi reálnymi komunitami považovali za správne komunity nájdené algoritmom CPM. Výsledky ostatných metód sme porovnávali potom vzhľadom k výsledkom CPM. Porovnávanie výsledkov prebiehalo za pomoci metrík popísaných ďalej v sekcii 6.1.

6.1 Metriky

Pri vyhodnotení zhody dvoch nájdených komunít budeme pracovať s metrikami *presnosť* P (*precision*) a *úplnosť* R (*recall*) [43]. Hodnota presnosti je daná ako pomer počtu správnych nových prvkov, ku všetkým novým prvkom a úplnosť je pomer počtu správnych nových prvkov, ku všetkých možným správnym prvkom.

Predstaviť si to môžeme jednoducho na nasledujúcom príklade.

Příklad 6.1
Majme komunitu $C = \{1, 2, 3, 4, 5, 6\}$ detekovanú metódou CPM, ktorú považujeme za správnu. Ďalej majme komunitu $I = \{4, 5, 6, 7, 8, 9, 10, 11, 12\}$ detekovanú metódou CIS. Chceme určiť hodnotu P a R komunity I vzhľadom na komunitu C.

- Komunita I sa „trafila“ v 3 vrcholov (4, 5, 6) z 9 (ktoré sama obsahuje), takže $P = 3/9$
- Komunita I „našla“ 3 vrcholy (4, 5, 6) zo 6 možných (ktoré obsahuje C), takže $R = 3/6$

■

Na presnosť sa môžeme pozerať ako na mieru kvality a na úplnosť ako na mieru kvantity. Jednoducho môžeme povedať, že veľká hodnota presnosti znamená, že algoritmus vracia viac relevantných výsledkov ako nerelevantných, zatiaľ čo vysoká hodnota úplnosti znamená, že algoritmus vrátil väčšinu z relevantných výsledkov.

Považujme komunitu C nájdenú algoritmom CPM za správnu a komunitu I nájdenú algoritmom CIS za novú, ktorej relevantnosť sa má vyhodnotiť. Potom môžeme zaviesť nasledujúce značenia :

- N - počet vrcholov v sieti
- M - počet vrcholov v komunite C
- TP (*true positive*) - počet vrcholov z komunity I, ktoré sú v komunite C (správne zaradené)
- FP (*false positive*) - počet vrcholov z komunity I, ktoré nie sú v komunite C (nesprávne zaradené)
- FN (*false negative*) - počet vrcholov z komunity C, ktoré nie sú v komunitě I (nesprávne vyradené)

- TN (*true negative*) - počet vrcholov, ktoré nie sú ani v komunite C a ani v komunite I (správne vyradené)

Presnosť P môžeme teraz definovať ako

$$P = \frac{TP}{TP + FP} \tag{21}$$

a úplnosť R ako

$$R = \frac{TP}{TP + FN}. \tag{22}$$

Ďalej definujeme *F-skóre* ($F_1 score$, *F-measure*), ktoré predstavuje harmonický priemer P a R, ako

$$F = 2 \times \frac{P \times R}{P + R}. \tag{23}$$

F-skóre nadobúda hodnoty od 0 po 1, pričom 1 znamená najlepšiu hodnotu.

Zavádzame tiež *Matthewsov korelačný koeficient* (MCC), ktorý je definovaný ako

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}. \tag{24}$$

MCC nadobúda hodnoty od -1 po 1, pričom 1 znamená najlepšiu hodnotu (predikciu), 0 náhodný výber a -1 najhoršiu hodnotu. Ak je vo vzťahu pre MCC v menovateli 0, tak je MCC definovaný ako 0.

F-skóre zvýhodňuje „pozitívne“ hodnoty, zatiaľ čo MCC nezvýhodňuje ani „negatívne“ ani „pozitívne“. F-skóre oproti MCC veľmi mierne zvýhodňuje presnosť a MCC naopak úplnosť [46]. Napr. pre :

- $TN = 850$, $FP = 50$, $FN = 35$, $TP = 65$, čo je $R = 0.65$ a $P = 0.5652$ dostávame $MCC = 0.559$ a $F = 0.605$
- $TN = 825$, $FP = 75$, $FN = 25$, $TP = 75$, čo je $R = 0.75$ a $P = 0.5$ dostávame $MCC = 0.560$ a $F = 0.6$ (drobné zníženie P spôsobilo drobné zníženie F-skóre)

Pri porovnávaní komunít je dôležitejšia presnosť ako úplnosť. Dôvod si ukážeme na nasledujúcom príklade.

Příklad 6.2
CPM odhalila v grafe $G(V, E)$, kde $V = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ dve komunity, $C_1 = \{1, 2, 3\}$ a $C_2 = \{7, 8, 9\}$. Tie považujeme za správne. Uvažujme ďalej nasledujúce dva prípady :

1. CIS odhalil dve komunity $I_1 = \{1, 2, 3, 4, 5\}$ a $I_2 = \{6, 7, 8, 9, 10\}$
2. CIS odhalil štyri komunity $I_1 = \{1, 2\}$, $I_2 = \{2, 3\}$, $I_3 = \{7, 8\}$, $I_4 = \{7, 9\}$

V prvom prípade je pre komunitu I_1 vzhľadom na C_1, $P = 3/5$ a $R = 1$ a pre komunitu I_2 vzhľadom na C_2 rovnako $P = 3/5$ a $R = 1$.

V druhom prípade je pre komunity I_1 a I_2 vzhľadom na C_1, $P = 1$ a $R = 2/3$ a pre komunity I_3 a I_4 vzhľadom na C_2 tiež $P = 1$ a $R = 2/3$. ■

Z príkladu usudzujeme, že hodnota presnosti je dôležitejšia, ako hodnota úplnosti (aj keď sa dá o tom určite polemizovať). Viac záleží na tom spojiť do 1 komunity len vrcholy, ktoré k sebe patria, ako spojiť do 1 komunity všetky vrcholy, ktoré k sebe patria, za cenu zahrnutia ďalších, ktoré k nim nepatria.

Mnoho malých komunít s vysokou presnosťou môže odhaľovať hierarchickú štruktúru vo vnútri pôvodnej komunity. Pri CIS mohli tiež vzniknúť kvôli tomu, že pri zlučovaní prekrývajúcich sa komunít sme nemuseli zlúčiť všetky, ktoré mali byť zlúčené.

Nemôžeme však tvrdiť, že je pre nás presnosť úplne smerodatná, pretože aj keď jej vysoká hodnota znamená, že sme do komunity zaradili vrcholy, ktoré k sebe patria, tak nám nič nevypovedá o samotnom počte komunít a ich veľkosti.

Z vyššie uvedených dôvodov sa budeme pri testoch riadiť hlavne mierou F-skóre, ktorá priemeruje presnosť a úplnosť a zároveň čiastočne zvýhodňuje presnosť.

Jednotlivé experimenty sú uvedené v nasledujúcich častiach. Ešte raz pripomíname, že u sietí, kde nepoznáme reálne komunity, považujeme za správne komunity nájdené metódou CPM.

6.2 Zacharyho karate klub

Štatistika siete sa nachádza v tabuľke 1.

Vrcholy	34
Hrany	78
Známe komunity	2

Tabuľka 1: Počet vrcholov a hrán - Zacharyho karate klub

Táto kolekcia predstavuje dobre známu sieť 34 členov Zacharyho karate klubu, ktorí boli pozorovaní v priebehu troch rokov [41]. Hrany v sieti spájajú jedincov, ktorí spolu „komunikovali“ aj mimo aktivít klubu. V klube nastal neskôr konflikt medzi prezidentom klubu (vrchol č. 34) a jedným z inštruktorov (vrchol č. 1), čo malo za následok rozdelenie členov klubu na dve skupiny, kde každá podporovala jedného z konfliktu.

Algoritmy na detekciu komunít sa snažia, len za pomoci znalosti štruktúry siete, tieto dve skupiny odhaliť.

6.2.1 Výsledky

Tabuľka 2 zobrazuje výsledky jednotlivých metód. Výsledky boli porovnávané vzhľadom na 2 komunity, ktoré vznikli rozpadom klubu. Nastavenia metód CIS, NCIS a GCIS sú uvedené v tabuľke 3.

V tabuľke 2 vidíme, že GCIS a S GCIS dávajú úplne rovnaké výsledky a NCIS a S NCIS takmer rovnaké. Zhodu základných verzií metód s ich variantom s urýchľovacou heuristikou sme testovali aj u rozsiahlejších sietí (napr. DBLP viď sekcia 6.4), kde vyšla taktiež veľmi vysoká zhoda výsledkov (0.95 avg. F-skóre). Na základe toho usudzujeme, že heuristika dáva vzhľadom na základnú verziu metódy veľmi presné výsledky.

	CPM	CIS	NCIS	GCIS	S GCIS	S NCIS	B CIS
po redukc.	33	-	-	-	-	-	-
komunity	3	2	2	3	3	2	2
min. kom.	3	19	19	5	5	17	19
max. kom.	25	29	19	19	19	19	19
avg. kom.	11	24	19	12	12	18	19
min. hust.	-	0.874	0.874	0.750	0.750	0.872	0.907
max. hust.	-	0.971	0.889	0.889	0.889	0.889	0.924
avg. hust.	-	0.922	0.881	0.809	0.809	0.880	0.915
pokrytie	94.12%	100%	100%	100%	100%	100%	100%
vrch v kom.	32	34	34	34	34	34	34
prekrýv. páry kom.	2	1	1	1	1	1	1
prekrýv. vrch.	2	14	4	4	4	2	4
avg. presnosť	0.853	0.741	0.895	0.917	0.917	0.918	0.895
avg. úplnosť	0.451	1	1	0.666	0.666	0.971	1
avg. F-skóre	0.496	0.842	0.944	0.724	0.724	0.943	0.944

Tabuľka 2: Výsledky detekcie komunít - Zacharyho karate klub

	Tuning param. λ	Similarity threshold
CIS, NCIS, GCIS	0	80%
S NCIS, S GCIS	0	80%
B CIS	0.3	60%

Tabuľka 3: Nastavenia metód založených na CIS - Zachary karate klub

Z výsledkov je ďalej vidieť, že aj keď by mala metóda CPM detekovať „najkvalitnejšie“ komunity, tak u tejto siete dosiahla CPM najmenšiu hodnotu F-skóre. Pokles F-skóre zapríčinili malé hodnoty úplnosti. Je to spôsobené najmä tým, že CPM nezaradila do žiadnej z komunít 2 vrcholy a tým, že oddelila 5 vrcholov do separátnej komunity (rovnako ako GCIS, viď obr. 29 - zelená komunita) a ďalšie 3 vrcholy taktiež. Tento fakt len potvrdzuje to, že niekedy (hlavne u sociálnych sietí) je veľmi ťažké správne definovať prepojenia medzi vrcholmi, tak aby reflektovali skutočnú situáciu z reálneho života.

Treba si všimnúť, že presnosť všetkých metód bola pomerne vysoká. Zhoršenie F-skóre zapríčiňili hlavne hodnoty úplnosti.

Najlepšie výsledky sme dostali s použitím NCIS v základnej (tiež v S verzii) a s CIS s upravenými parametrami.

Časy detekcie sa pre všetky CIS metódy a CPM pohybovali pod *0.005 s*, pri použití urýchľovacej heuristiky bol čas zanedbateľný.

Väčšina metód v testoch označila, že vrcholy 3, 9, 10 a 31 patria do oboch komunít. Aj keď sa tieto vrcholy v skutočnosti zaradili len do jednej komunity, tak prekrytie nám môže naznačovať, že ľudia reprezentovaní týmito vrcholmi nemali rozhodovanie ľahké.

Tento fakt je pekne vidieť aj na vizualizácii siete na obr. 28.

Pre zaujímavosť uvádzame aj porovnanie výsledkov CIS metód vzhľadom na výsledky CPM, viď tabuľka 4. Treba mať na pamäti, že CPM odhaľovalo komunity o min. veľkosti 3, zatiaľ čo ostatné o min. veľkosti 1.

	avg. F-skóre	avg. presnosť	avg. úplnosť
CIS	0.636	0.737	0.560
NCIS	0.636	0.737	0.560
GCIS	0.750	0.900	0.651

Tabuľka 4: Porovnanie výsledkov metód s CPM - Zacharyho karate klub

Všimnime si, že aj keď GCIS mala vzhľadom na reálne komunity menšiu zhodu (hodnotu F-skóre) ako CIS a NCIS, tak vzhľadom na komunity nájdené CPM je to práve opačne. To naznačuje, že metóda GCIS by mohla detekovať s defaultnými nastaveniami ($\lambda = 0$) štrukturálne „kvalitnejšie“ komunity, ako CIS a NCIS.

Vizualizácie výsledkov metód sú uvedené v prílohe B.

6.3 Delfíni

Štatistika siete sa nachádza v tabuľke 5.

Vrcholy	62
Hrany	159
Známe komunity	2

Tabuľka 5: Počet vrcholov a hrán - Delfíni

Kolekcia predstavuje sieť delfínov skákavých (z angl. „bottlenose“) žijúcich pri Novom Zélande (Doubtful Sound) [42]. V sieti je 62 delfínov, pričom hrany boli zaznačené medzi delfíny, ktoré boli spolu v rozpätí siedmych rokov (1994-2001) videné častejšie ako sa očakávalo. Delfíni sa rozdelili na 2 veľmi dobre rozlíšitelné komunity (spojené len 6 hranami) samcov a samíc. Vďaka tomu sa táto sieť často používa na testovanie algoritmov detekcie komunít.

6.3.1 Výsledky

Výsledky jednotlivých metód sú zobrazené v tabuľke 6. Výsledky boli porovnávané vzhľadom na skutočné rozdelenie delfínov do 2 komunít. Nastavenia metód CIS, NCIS a GCIS sú uvedené v tabuľke 7.

Z výsledkov vidíme, že CPM má zo všetkých metód najväčšiu hodnotu presnosti, no rovnako, ako v prípade Zacharyho karate klubu (viď tabuľka 2), má veľmi malú hodnotu úplnosti. CPM v podstate správne zaraďuje takmer všetky vrcholy, no vytvorené

	CPM	CIS	NCIS	GCIS	S GCIS	S NCIS	B GCIS	B NCIS
po redukc.	53	-	-	-	-	-	-	-
komunity	4	3	2	4	4	4	2	2
min. kom.	5	10	21	7	7	7	23	23
max. kom.	25	48	50	42	42	42	42	44
avg. kom.	13	26	35	18	18	19	32	33
min. hust.	-	0.680	0.936	0.667	0.667	0.667	0.923	0.934
max. hust.	-	0.969	0.962	0.973	0.973	0.973	0.973	0.977
avg. hust.	-	0.862	0.949	0.800	0.800	0.812	0.948	0.956
pokrytie	74.19%	100%	100%	100%	100%	100%	100%	100%
vrch. v kom.	46	62	62	62	62	62	62	62
prekrýv. kom.	3	3	1	5	5	5	1	1
prekrýv. vrch.	6	11	9	12	12	17	3	5
avg. presnosť	0.981	0.876	0.896	0.933	0.933	0.867	0.935	0.913
avg. úplnosť	0.415	0.800	1.000	0.613	0.613	0.613	1.000	1.000
avg. F-skóre	0.557	0.814	0.945	0.713	0.713	0.694	0.965	0.954

Tabuľka 6: Výsledky detekcie komunít - Delfíni

	Tuning param. λ	Similarity threshold
CIS, NCIS, GCIS	0	80%
S NCIS, S GCIS	0	80%
B NCIS	0.115	60%
B GCIS	0	35%

Tabuľka 7: Nastavenia metód založených na CIS - Delfíni

komunity sú neúplné (bolo by ich potrebné zlúčiť). Rovnakým problémom trpí aj metóda GCIS (vo všetkých variantoch), ktorá vďaka greedy optimalizácii zaraďuje vrcholy veľmi presne, no vzniknuté komunity sú tiež neúplné. Tento problém môže byť vyriešený efektívnejším post processingom komunít, vďaka čomu by sa z GCIS stala takmer ideálna metóda („takmer“ kvôli väčšej časovej náročnosti oproti NCIS).

Najlepšie výsledky v defaultnej konfigurácii (bez S heuristiky) dostávame pomocou metód CIS a NCIS, ktoré majú vyvážené hodnoty presnosti a úplnosti. Aj napriek tomu ale detekuje CIS veľké množstvo prekrývajúcich sa vrcholov, ktoré sa v reále neprekrývajú.

Pri upravenej konfigurácii sa nám najlepšie výsledky podarilo získať pomocou GCIS, kde 1 komunita bola odhalená úplne presne a do 2. boli zaradené tri vrcholy navyše (28, 30, 39). Tieto 3 vrcholy však boli označené ako prekrývajúce sa a z vizualizácie siete v prílohe B na obr. 34 je vidno, že sa jedná naozaj o hraničné vrcholy, ktoré sa vzhľadom na štrukturálne vlastnosti nedajú jednoznačne zaradiť.

6.4 DBLP

Štatistika siete sa nachádza v tabuľke 8.

Vrcholy	52615
Hrany	65000
Známe komunity	-

Tabuľka 8: Počet vrcholov a hrán - DBLP

Jedná sa o citačnú sieť bibliografie z oblasti počítačovej vedy. Dáta sú z Decembra 2012. Vrcholy predstavujú autorov článkov a hrany medzi autormi znamenajú, že spolu publikovali aspoň 1 publikáciu. Sieť má po určitej redukcii 89096 vrcholov a 65000 hrán, no mnoho z týchto vrcholov je izolovaných. Izolované vrcholy sme odstránili, čím sme dostali sieť o 52615 vrcholoch a 65000 hranách.

Keďže neexistujú definované reálne komunity, tak budeme výsledky metód porovnávať vzhľadom na výsledky metódy CPM ($k = 3$). Preto sú v tabuľke 9 zobrazené výsledky, kde sme počítali s min. veľkosťou komunity 3. V tejto kolekcii okrem nami implementovaných metód uvedieme porovnanie aj pre výsledky metódy VDM [14].

6.4.1 Výsledky

Tabuľka 9 zobrazuje výsledky jednotlivých metód. Nastavenia metód CIS, NCIS a GCIS sú uvedené v tabuľke 10.

Nepotvrdil sa nám predpoklad z testov Zacharyho karate klubu o tom, že GCIS dáva štrukturálne lepšie komunity ako NCIS (rozdiel F-skóre S NCIS a S GCIS0 je tu len 0.006).

Z tabuľky 9 je vidieť, že CPM zredukovalo sieť približne o polovicu, preto je pokrytie grafu komunitami len 50.8%. Táto sieť je pomerne riedko prepojená, čo v spojení s prísnou definíciou komunity podľa CPM spôsobuje takéto malé pokrytie. Naopak u základných verzií (S) CIS je pokrytie, aj napriek požiadavku min. veľkosti komunity 3, pomerne veľké - 80%.

Tiež je vidieť, že VDM s GCIS1 dávajú podobné výsledky, aj keď VDM má väčšiu presnosť, no zato menšiu úplnosť.

Veľmi zaujímavá je konfigurácia najleších (B) CIS metód. Je zrejmé, že pri nastavení veľkej hodnoty tuning parametra λ sa výsledné komunity veľmi podobajú na kliky. Čím väčší je parameter lambda, tým viac eliminujeme vrcholy, ktoré pri pripojení do komunity majú menší počet spojení v komunite ako zvyšné vrcholy komunity. Tým pádom vytvárame štruktúry veľmi podobné klikám. Preto je aj výsledná presnosť a F-skóre pre tieto metódy vzhľadom na CPM veľmi vysoká. Dá sa povedať, že sú takmer zhodné s výsledkami CPM, aj čo sa týka pokrytia grafu.

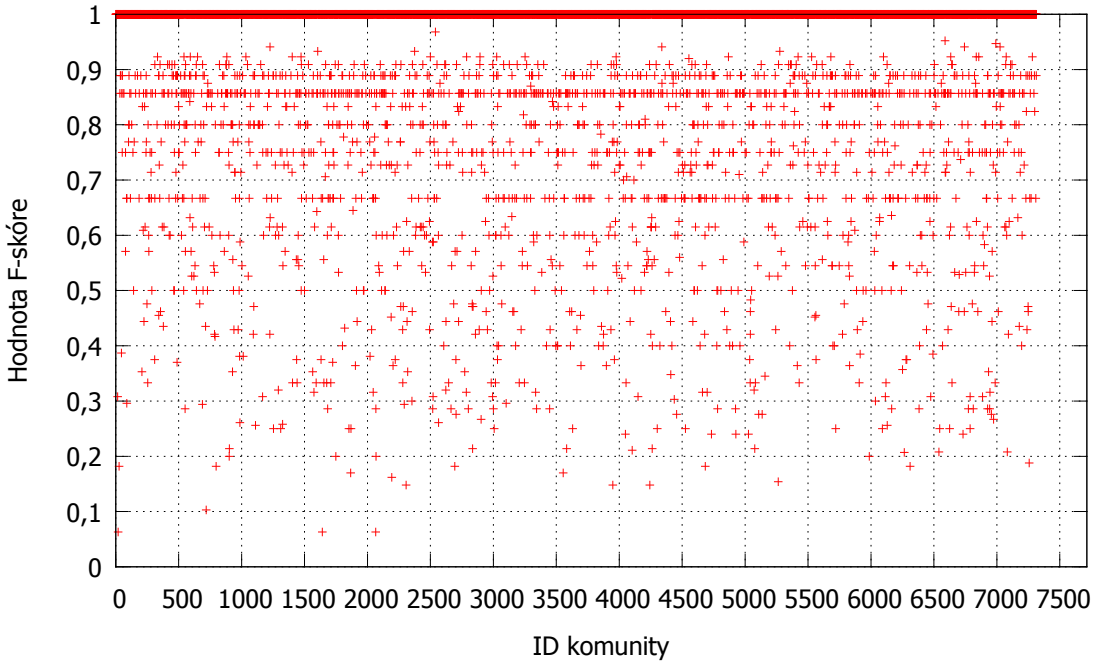
Rozdelenie F-skóre jednotlivých komunít nájdených s BS GCIS je znázornené v grafe na obr. 25 a kumulatívna distribúcia F-skóre je ukázaná v grafe na obr.26. Grafy pre ostatné metódy sú uvedené v prílohe C.

	CPM	CIS	S NCIS	S GCIS0	S GCIS1	VDM	BS GCIS	BS NCIS
po redukc.	27189	-	-	-	-	-	-	-
komunity	7528	7078	7715	8101	9656	9644	7315	7273
min. kom.	3	3	3	3	3	3	3	3
max. kom.	92	96	63	37	28	37	29	28
avg. kom.	3	6	5	5	3	-	3	3
min. hust.	-	0.400	0.375	0.364	0.597	-	0.992	0.992
max. hust.	-	1.000	1.000	1.000	1.500	-	2.000	2.000
avg. hust.	-	0.958	0.938	0.925	1.223	-	1.716	1.719
pokrytie	50.80%	80.18%	80.08%	79.22%	63.63%	-	46.20%	46.02%
vrch. v kom.	26729	42188	42136	41681	33478	-	24309	24216
prekrýv. kom.	4100	1325	1011	764	2190	-	2015	1930
prekrýv. vrch.	2266	3301	1873	1217	2145	-	1884	1835
avg. presnosť	base	0.533	0.547	0.560	0.743	0.990	0.996	0.996
avg. úplnosť	base	0.733	0.722	0.715	0.704	0.625	0.899	0.898
avg. F-skóre	base	0.594	0.603	0.609	0.711	0.750	0.929	0.929

Tabuľka 9: Výsledky detekcie komunít - DBLP

	Tuning param. λ	Similarity threshold
CIS	0	60%
S NCIS, S GCIS0	0	80%
S GCIS1	1	80%
BS GCIS, BS NCIS	2	60%

Tabuľka 10: Nastavenia metód založených na CIS - DBLP



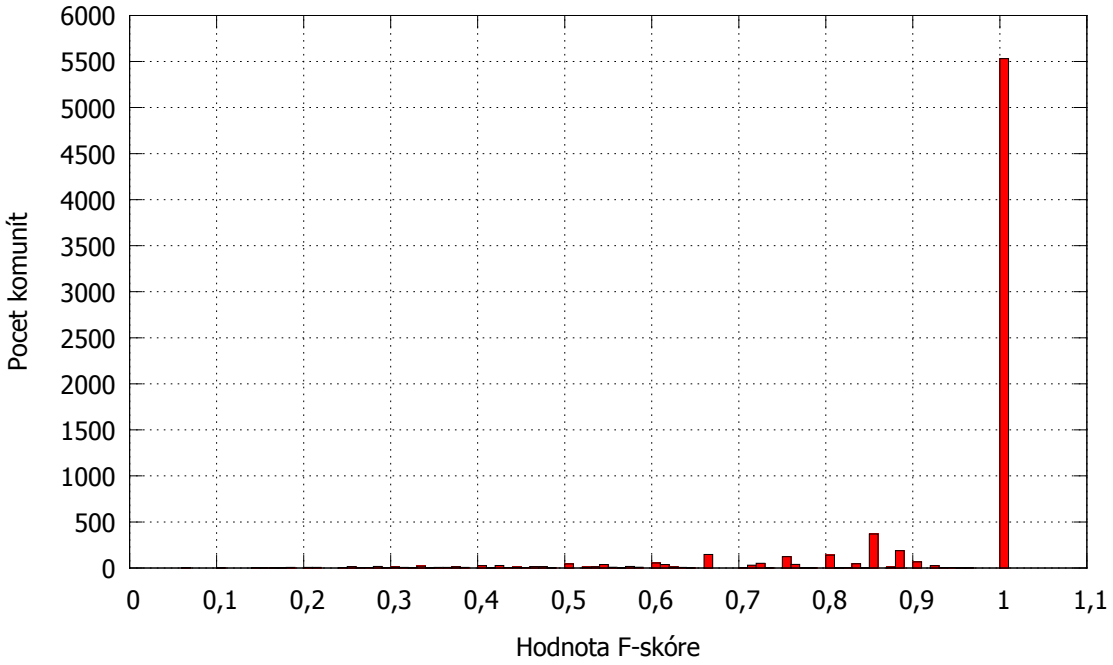
Obr. 25: Zobrazenie F-skóre vzhľadom na jednotlivé komunity - BS GCIS

V prvých dvoch experimentoch sa ukázalo, že samotná metóda CPM nedetekuje komunity odpovedajúce reálnemu svetu, resp. neodhalí ich všetky kvôli svojej prísnej definícii komunity. Preto vysoká zhoda CIS metód (v určitej konfigurácii) s výsledkami CPM nezaručuje tiež zhodu s reálnymi komunitami. V prvých dvoch testoch sme si však ukázali, že CIS metódy dokážu detekovať reálne komunity veľmi presne.

Tento fakt je veľmi dôležitý, pretože poukazuje na široké možnosti CIS metód. Pri vhodných parametroch dokážu detekovať riedke aj husté komunity, alebo len veľmi husté (takmer kliky). Jednoduchou modifikáciou parametrov ich potom môžeme použiť na rôzne typy sietí.

Výhodná je tiež časová náročnosť metód S NCIS a S GCIS, viď tabuľka 11.

Potvrdil sa predpoklad, že CIS je vo svojej základnej verzii časovo veľmi náročná. Naopak nami vytvorené metódy NCIS a GCIS sú v kombinácii s S heuristikou veľmi rýchle,



Obr. 26: Kumulatívna distribúcia F-skóre jednotlivých komunít - BS GCIS

	vyhľadanie komunít / s	post processing / s	počet vlákien
CIS	1035.339	21.843	-
GCIS	3.959	10.060	-
NCIS	1.861	19.784	-
paralel. CIS	308.882	19.404	8
paralel. GCIS	0.915	10.135	8
paralel. NCIS	0.457	19.141	8
S GCIS	0.699	11.547	-
S NCIS	0.366	13.588	-

Tabuľka 11: Tabuľka časov metód založených na CIS - DBLP

dokonca rýchlejšie ako paralelné spracovanie. Vzhľadom na počet nájdených komunít je časovo náročný len post processing komunít, ktorý by mohol byť vylepšený.

Ďalej uvádzame grafy pre metódy CIS, S NCIS, S GCIS a VDM, ktoré zobrazujú rôzne distribúcie a informácie o kvalite detekovaných komunít vzhľadom na CPM.

6.5 Enron

Štatistika siete sa nachádza v tabuľke 12.

Vrcholy	36692
Hrany	367662
Známe komunity	-

Tabuľka 12: Počet vrcholov a hrán - Enron

Táto sieť predstavuje emailovú komunikáciu zamestnancov firmy Enron. Jednalo sa o americkú energetickú spoločnosť, ktorá dlhé roky prosperovala a bola vzorom podnikania, no po odhalení podvodov v účtovníctve vyhlásila v roku 2002 bankrot, po ktorom nasledovali právne spory s vedením spoločnosti.

Dáta boli zverejnené počas vyšetrovania podvodov v spoločnosti. Vrcholy siete predstavujú emailové adresy zamestnancov a hrany znamenajú, že z jednej adresy na druhú bol odoslaný aspoň 1 email [52].

V d'alšej časti nebudeme uvádzať porovnania vzhľadom na reálne komunity, pretože nie sú známe a ani vzhľadom na CPM, pretože sa nám s ňou sieť nepodarilo spracovať (pre $k = 3$). Pokusy sa budú týkať hlavne časov trvania metód CIS a vzájomného porovnania ich výsledkov.

6.5.1 Výsledky

Počas spracovávania tejto siete sme narazili na problém nedostatočnej efektivity implementácie clique clique overlap matice v metóde CPM. Algoritmus vyhľadávania klík Bron-Kerbosch trval len 34 s, no kvôli tomu, že je táto sieť veľmi hustá a obsahuje veľké množstvo klík, sa nepodarilo maticu kvôli nedostatku pamäte zostrojiť (ani po zvýšení pamäte pre Javu). Vytváranie matice by teda mohlo byť ešte viac optimalizované, prípadne úplne preskočené (pri prechádzaní klík by sa priamo vytvárali komunity). CPM v aktuálnej podobe sa teda nehodí používať na veľmi husté siete.

Sieť sme kvôli jej hustote zvyšujúcej časovú náročnosť výpočtu, analyzovali len s metódami NCIS a GCIS (CIS mala problémy už s DBLP, viď tabuľka 11). Nasledujúca tabuľka 13 zobrazuje výsledky metód, ktoré sú porovnané vzhľadom na paralelnú verziu NCIS. Nastavenia boli pre všetky metódy rovnaké : similarity threshold - 60%, tuning param. $\lambda = 0$.

Z výsledkov je vidieť, že podobnosť výsledkov je pre NCIS a S NCIS aj pre takto hustú sieť pomerne vysoká (F-skóre - 0.864). Opäť sa ukázalo, že GCIS dosiahlo pri porovnávaní veľmi vysokú priemernú presnosť (0.963), no problémom ostáva nízka priemerná hodnota úplnosti (0.503).

V tabuľke s časmy sa jasne ukazuje problém metódy GCIS s veľmi hustými sieťami. Keďže každý uzol má veľký počet susedov tak sa optimalizácia pridávaného vrcholu v skene spomaľuje. S NCIS bola približne 15x rýchlejšia ako GCIS a takmer 4x rýchlejšia ako paralelná verzia NCIS. Zaujímavé však je, že post processing vrcholov trval u S GCIS len 3.5 s.

Výkonnosť S NCIS budeme demonštrovať aj v nasledujúcom teste.

	NCIS	S NCIS	S GCIS
komunity	1884	2003	2817
min. kom.	2	2	2
max. kom.	3536	2653	1231
avg. kom.	59	37	10
min. hust.	0.287	0.323	0.336
max. hust.	1.000	1.000	1.000
avg. hust.	0.801	0.797	0.761
pokrytie	96.64%	95.67%	74.93%
vrch. v kom.	35459	35105	27495
prekrýv. kom.	39299	21767	1220
prekrýv. vrch.	20753	14035	2469
avg. presnosť	base	0.964	0.963
avg. úplnosť	base	0.840	0.503
avg. F-skóre	base	0.864	0.540

Tabuľka 13: Výsledky detekcie komunít - Enron

	vyhľadanie komunít/min	post processing/min	poč. vlákien
paralel. NCIS	15.45	5.88	8
S GCIS	62.37	0.06	-
S NCIS	4.01	1.66	-

Tabuľka 14: Tabuľka časov metód založených na CIS - Enron

6.6 Sieť produktov z Amazon.com

Štatistika siete sa nachádza v tabuľke 15.

Vrcholy	400 727
Hrany orientované	3 200 440
Hrany neorientované	2 349 869
Známe komunity	-

Tabuľka 15: Počet vrcholov a hrán - Enron

Táto sieť bola vytvorená automatizovaným prechádzaním stránky Amazon. Je založená na funkcii Amazon stránky : „Kto si kúpil tento tovar, tiež kúpil aj ...“. Ak je produkt často kupovaný s iným produktom, tak graf obsahuje medzi nimi orientovanú hranu.

Dáta pochádzajú z 12 Marca 2003 [53]. Keďže sa jedná o orientovanú sieť, tak sme ju najskôr previedli na neorientovanú.

S NCIS dokázalo túto sieť spracovať za 24 min., problémom však bol pomalý post processing výsledkov, ktorý trval až 4.5 hod.

Pre úplnosť uvádzame v tabuľke výsledky S NCIS metódy.

	S NCIS
komunity	20915
min. kom.	2
max. kom.	1874
avg. kom.	37
min. hust.	0.317
max. hust.	0.998
avg. hust.	0.605
pokrytie	97.63%
vrch. v kom.	391229
prekrýv. kom.	230138
prekrýv. vrch.	196450

Tabuľka 16: Výsledky detekcie komunít - Amazon

7 Záver

V práci sme uviedli čitateľa do problematiky komplexných sietí a ich analýzy. Následne sme sa venovali popisu detekcie komunít v komplexných sieťach a prehľadu jednotlivých metód. Podrobnejšie sme sa zaoberali popisom lokálnych metód detekujúcich prekrývajúce sa komunity. Ďalej sme detailne popísali metódy CPM a CIS, ktoré sme ako súčasť práce naimplementovali. Počas analýzy sme zistili, že väčšina lokálnych prístupov si je veľmi podobná. Líšia sa len v drobných modifikáciach funkcie hustoty, postupe generovania seed skupín a spracovania týchto skupín. Avšak aj takéto, na prvý pohľad, drobné modifikácie dokážu veľmi ovplyvniť výsledky a rýchlosť ich spracovania.

Na základe poznatkov získaných počas analýzy iných metód sme zaviedli dve úpravy CIS, ktoré prispeli k výraznému zvýšeniu efektivity tejto metódy. Z úprav vznikli dve nové metódy, ktoré sme v rámci práce nazvali NCIS (Neighbors CIS) a GCIS (Greedy CIS). Ku všetkým CIS metódam sme vytvorili aj paralelné verzie.

Aplikácia, ktorá vznikla ako súčasť tejto práce obsahuje implementáciu spomínaných metód s možnosťou exportu detailných štatistík vrátane výpisu jednotlivých prekrývajúcich sa komunít a ich vrcholov. Okrem toho umožňuje tiež export analyzovaného grafu do formátu .gexf s farebne odlíšenými komunitami a zvýraznenými prekrývajúcimi sa vrcholmi. V neposlednom rade umožňuje porovnávať výsledky jednotlivých metód na základe rôznych metrík.

V experimentálnej časti práce sme vykonali niekoľko experimentov nad reálnymi sieťami, ale aj nad testovacími grafmi, ktoré sme tu pre stručnosť neuviedli. V experimentoch sme zistili, že metóda CPM kvôli pameťovým nárokom pri analýze väčších, hustejších sietí zlyháva. Najslabším článkom tejto metódy je konštrukcia matice prekrytia klík. Aj keď sme sa snažili o jej efektívnu implementáciu za pomoci riedkej symetrickej matice, tak táto fáza algoritmu môže byť do budúcna ešte viac optimalizovaná.

Ďalej sa v prípade CPM potvrdilo, že jej prísna definícia komunity (spojenie k-klík) síce zabezpečí vysokú úroveň *presnosti* zaradenia vrcholov do komunít, no v rovnakom čase zlyháva pri dodržaní *úplnosti*. Stačí, aby husto prepojenej komunite chýbala len 1 hrana na splnenie podmienky klikovosti a už ju CPM nepovažuje za komunitu. Veľké množstvo vrcholov nepriradí do žiadnej komunity. Táto metóda je teda vhodná na použitie pre husto prepojené grafy menšej veľkosti.

Pri testovaní CIS metód sme si potvrdili predpoklad, že CIS je pre väčšie siete časovo veľmi náročná. Zároveň sme overili, že nami vytvorené metódy NCIS a GCIS sú naopak veľmi rýchle. Ukázalo sa, že aj s použitím urýchľovacej heuristiky dávajú metódy NCIS a GCIS takmer rovnaké výsledky ako bez nej, pričom sa súčasne zvyšuje výkonnosť až tak, že sú časovo menej náročné ako ich paralelné verzie bez tejto heuristiky.

Pri pokusoch sme si všimli, že GCIS sa pri vysokej hodnote parametra λ správa takmer ako CPM. Vytvára množstvo husto prepojených komunít podobných klikám. Vďaka tomu, že výsledné komunity ale nemusia byť tvorené klikami pokrýva väčšiu časť grafu ako CPM. Ukázalo sa teda, že jednoduchou modifikáciou parametrov môžeme GCIS efektívne použiť na rôzne typy sietí s rôznou štruktúrou.

S metódou NCIS sa nám podarilo spracovať v rozumnom čase (rádovo desiatky minút) aj pomerne veľké siete (sieť produktov Amazon mala cez 400 tisíc vrcholov a 2.3

milióna hrán). Ukázalo sa, že metóda NCIS je zo všetkých implementovaných metód časovo najefektívnejšia a zároveň dáva vyvážené hodnoty presnosti a úplnosti.

Za hlavnú slabinu nami implementovanej metódy CIS (rovnako aj GCIS a NCIS) môžeme považovať fázu post-processingu výsledných komunít, ktorá môže byť vylepšená o efektívnejšie zlučovanie komunít, prípadne zavedenie vhodnejšej miery podobnosti. Rovnako môže byť zaujímavé porovnať a implementovať rôzne spôsoby prípravy seed skupín. Pri implementácii sme sa zaoberali len neohodnotenými neorientovanými grafmi, takže práca môže byť rozšírená aj o variácie metód pre ohodnotené grafy.

Zistili sme, že aplikácia algoritmov silno závisí od štruktúry analyzovanej siete. Nedá sa teda povedať, ktorý algoritmus je jednoznačne najlepší, pretože každý ma svoje pre a proti.

Počas práce sme si veľmi rozšírili poznatky z teórie grafov a to hlavne v oblasti detekcie komunít. V práci sme uviedli prehľadný popis lokálnych metód a implementovali sme požadované algoritmy vo forme, v ktorej zvládajú spracovať aj veľmi veľké grafy, rádovo so stovkami tisíc vrcholov a miliónmi hrán. Prácu z našej strany, z týchto dôvodov považujeme za vydarenú a pre nás veľmi prínosnú.

Na záver môžeme len konštatovať, že problém detekcie komunít nemá, aj napriek veľkej snahe vedcov, uspokojivé riešenie.

Martin Gajdičiar

8 Literatúra

[1] NEWMAN, M.E.J, *The structure and function of complex networks*, SIAM Review 45, s. 167-256, 2003.

[2] FORTUNATO, S., *Community detection in graphs*, Physics Reports 486, s. 75-174, 2010.

[3] PALLA, G. - DERÉNYI, I. - FARKAS, I. - VICSEK, T., *Uncovering the overlapping community structure of complex networks in nature and society*, Nature 435, s. 814-818, 2005.

[4] PALLA, G. - DERÉNYI, I. - FARKAS, I. - VICSEK, T., *Uncovering the overlapping community structure of complex networks in nature and society - Supplementary Material*, Nature 435, no. 7043, pp. 814-818, 2005.

[5] BOMZE, I. M. - BUDINICH, M. - PARDALOS, M. - PELILLO, M., *The maximum clique problem*, in D.-Z. Du, P. Pardalos (Eds.), Handbook of Combinatorial Optimization, Kluwer Academic Publishers, Norwell, USA, s. 1-74, 1999.

[6] BRON, C. - KERBOSCH, J., *Algorithm 457: finding all cliques of an undirected graph*, Commun. ACM, vol. 16, no. 9, pp. 575–577, 1973.

[7] CAZALS, F. - KARANDE, C., *A note on the problem of reporting maximal cliques*, Theoretical Computer Science, 407(1-3): s. 564 – 568, 2008.

[8] KELLEY, S. - GOLDBERG, M. - MAGDON-ISMAIL, M. - MERTSALOV, K. - WALLACE, A., Defining and Discovering Communities in Social Networks, In *Handbook of Optimization in Complex Networks: Theory and Applications*, Springer Publisher, kap. 6, s. 139-168, 2011.

[9] XIE, J. - KELLEY, S. - SZYMANSKI, K. B., *Overlapping Community Detection in Networks: The State of the Art and Comparative Study*, ACM Computing Surveys (CSUR), vol. 45, issue 4, Article No. 43 ,2013.

[10] BAUMES, J. - GOLDBERG, M. K. - KRISHNAMOORTHY, M. S. - MAGDON-ISMAIL, M. - PRESTON, N., *Finding communities by clustering a graph into overlapping subgraphs*, ADIS AC, s. 97-104. IADIS, 2005.

[11] BAUMES, J. - GOLDBERG, M. S. - MAGDON-ISMAIL, M., *Efficient identification of overlapping communities* in IEEE International Conference on Intelligence and Security Informatics, ISI, s. 27-36 2005.

[12] LANCICHINETTI, A. - FORTUNATO, S. - KERTÉSZ, J., *Detecting the overlapping and hierarchical community structure of complex networks*, New J. Phys. 11, 033015, 2009.

[13] RADICCHI, F. - CASTELLANO, C. - CECCONI, F. - LORETO, V. - PARISI, D., *Defining and identifying communities in networks*, Proc. Natl. Acad. Sci. USA, 101(9):2658–2663, 2004.

[14] KUDĚLKA, M. - DRÁŽDILOVÁ, P. - OCHODKOVÁ, E. - SLANINOVÁ, K. - HORÁK, Z., *Local community detection and visualization: Experiment based on student data*, in Proceedings of the Third International Conference on Intelligent Human Computer Interaction (IHCI 2011), Prague, Czech Republic, Springer, s. 291–303, 2011.

[15] ZEHNALOVA, S. - KUDĚLKA, M. Jr. - KUDĚLKA, M. - SNÁŠEL, V., *Comparing Two Local Methods for Community Detection in Social Networks*, in Computational Aspects of Social Networks (CASoN), s.155 - 160, 21-23 Nov. 2012.

[16] HAVEMANN, F. - HEINZ, M. - STRUCK, A. - GLASER, J., *Identification of overlapping communities and their hierarchy by locally calculating community-changing resolution levels*, J. Stat. Mech., 01, P01023, 2011.

[17] LANCICHINETTI, A. - RADICCHI, F. - RAMASCO, J. J. - FORTUNATO, S., *Finding statistically significant communities in networks*, PLoS ONE 6, 4, e18961, 2011.

[18] MOLLOY, M. - REED, B., *A critical point for random graphs with a given degree sequence*, Rand. Struct. Algo. 6, s. 161–179 1995.

[19] SHEN, H. - CHENG, X. - CAI, K. - HU, M.-B., *Detect overlapping and hierarchical community structure*, Physica A 388, 1706., 2009.

[20] LEE, C. - REID, F. - MCDAID, A. - HURLEY, N., *Detecting highly overlapping community structure by greedy clique expansion*, In Proc. SNAKDD Workshop. s. 33–42, 2010.

[21] GIRVAN, M. - NEWMAN, M. E. J., *Community structure in social and biological networks*, Proc Natl Acad Sci USA, 99(12):7821–6, 2002.

[22] NEWMAN, M., *Detecting community structure in networks*, The European Physical Journal B-Condensed Matter and Complex Systems, vol. 38, no. 2, s. 321–330, 2004.

[23] NEWMAN, M. E, *Modularity and community structure in networks*, Proc. Natl. Acad. Sci. USA, 103(23):8577–8582, 2006.

[24] CLAUSET, A. - MOORE, C. - NEWMAN, M. E. J., *Finding community structure in very large networks*, Physical Review E, 70(6):066111, 2004.

[25] GUIMERÁ, R. - SALES-PARDO, M. - AMARAL, L. A. N., *Modularity from fluctuations in random graphs and complex networks*, Physical Review E, 70(2):025101, 2004.

[26] WAKITA, K. - TSURUMI, T. - AMARAL, L. A. N., *Finding community structure in mega-scale social networks*, Arxiv preprint cs/0702048, 2007.

[27] PAGE, L., et al., *The pagerank citation ranking: Bringing order to the web*, Stanford Digital Libraries Working Paper, 1998.

[28] WILKINSON, D.M. - HUBERMAN, B.A., *A method for finding communities of related genes*, Proc. Natl. Acad. Sci. U.S.A. 101, 2004.

[29] MILGRAM, S., *The small world problem*, Psychology Today 2, s. 60–67, 1967.

[30] TRAVERS J. - MILGRAM, S., *An experimental study of the small world problem*, Sociometry 32, s. 425–443, 1969.

[31] EULER, L., *Solutio problematis ad geometriam situs pertinentis*, Commentarii Academiae Petropolitanae 8, 1736.

[32] ALBERT, R. - JEONG, H. - BARABÁSI, A. L., *Diameter of the world-wide web*, Nature 401, s. 130–131, 1999.

[33] ALBERT, R. - JEONG, H. - BARABÁSI, A. L., *Scale-free characteristics of random networks: The topology of the World Wide Web*, Physica A 281, s. 69–77, 2000.

[34] WATTS, D. J., *Small Worlds*, Princeton University Press,Princeton, 1999.

[35] AMARAL, L. A. N. - SCALA, A. - BARTHÉLÉMY, M. - STANLEY, H. E., *Classes of small-world networks*, Proc. Natl. Acad. Sci. USA 97, 11149–11152, 2000.

[36] LATORA, V. - MARCHIORI, M., *Is the Boston subway a small-world network?*, Physica A 314, s. 109–113, 2002.

[37] FALOUTSOS, M. - FALOUTSOS, P. - FALOUTSOS, C., *On power-law relationships of the internet topology*, Computer Communications Review 29, s. 251–262, 1999.

[38] JEONG, H. - TOMBOR, B. - ALBERT, R. - OLTVAI, Z. N. - BARABÁSI, A. L., *The large-scale organization of metabolic networks*, Nature 407, s. 651–654, 2000.

[39] FELL, D. A. - WAGNER, A., *The small world of metabolism*, Nature Biotechnology 18, 1121–1122, 2000.

[40] MAGNUSSON, J. - JONSSON, H. - SCHNURER, J. - ROOS, S., *Weissella soli sp. nov., a lactic acid bacterium isolated from soil*, International Journal of Systematic and Evolutionary Microbiology vol. 52, part 3, s. 831-834, 2002.

[41] ZACHARY, W. W., *An information flow model for conflict and fission in small groups*, Journal of Anthropological Research 33, s. 452-473, 1977.

[42] LUSSEAU, D. - SCHNEIDER, K. - BOISSEAU, O. J. - HAASE, P. - SLOOTEN, E. - DAWSON S. M., *The bottlenose dolphin community of Doubtful Sound features a large proportion of long-lasting associations*, Behavioral Ecology and Sociobiology 54, s. 396-405, 2003.

[43] MAKHOUL, J. - KUBALA, F. - SCHWARTZ, R. - WEISCHEDEL, R., *Performance measures for information extraction*, in Proceedings of DARPA Broadcast News Workshop, Herndon, VA, 1999.

[44] KOVÁŘ, Petr, *Teorie grafů* [online]. c2013, [cit. 2014-04-11].URL: <http://homel.vsb.cz/~kov16/files/skriptum_teorie_grafu_rozsirene.pdf>

[45] VEČERKA, Arnošt, *Grafy a grafové algoritmy* [online]. c2007, [cit. 2014-04-11].URL: <http://phoenix.inf.upol.cz/esf/ucebni/Grafy_a_grafove_algoritmy.pdf>

[46] Amrinder Arora, *Matthew's Correlation Coefficient – How Well Does It Do?* [online]. 2011, [cit. 2014-04-23].URL:

<http://standardwisdom.com/softwarejournal/2011/12/matthews-correlation-coefficient-how-well-does-it-do/>

[47] Trove, *Trove* [online]. c2008, [cit. 2014-04-17].URL:

<http://trove.starlight-systems.com/>

[48] HEYMANN, Sebastien, *GEXF File Format* [online]. c2012, [cit. 2014-04-19].URL:

<http://gexf.net/format/index.html>

[49] Gephi, *Gephi, an open source graph visualization and manipulation software* [online]. c2008-2012, [cit. 2014-04-19].URL:

<https://gephi.org/>

[50] FICAROLA, Francesco, *gexf4j - A java library for the GEXF file format* [online]. c2012, [cit. 2014-04-19].URL:

<https://github.com/francesco-ficarola/gexf4j>

[51] CFinder, *[CFinder] Clusters and Communities: Overlapping dense groups in networks* [online]., [cit. 2014-04-22].URL:

<http://cfinder.org/>

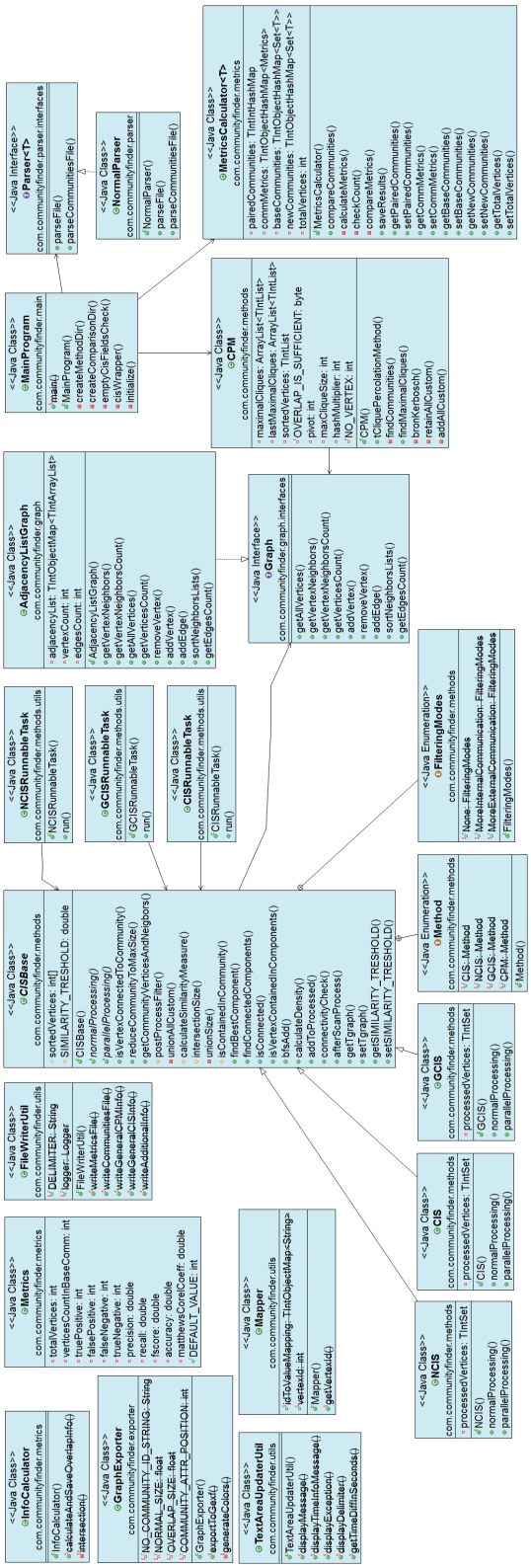
[52] LESKOVEC, Jure, *Enron email network* [online]., [cit. 2014-04-25].URL:

<http://snap.stanford.edu/data/email-Enron.html>

[53] LESKOVEC, Jure, *Amazon product co-purchasing network, March 12 2003* [online]., [cit. 2014-04-25].URL:

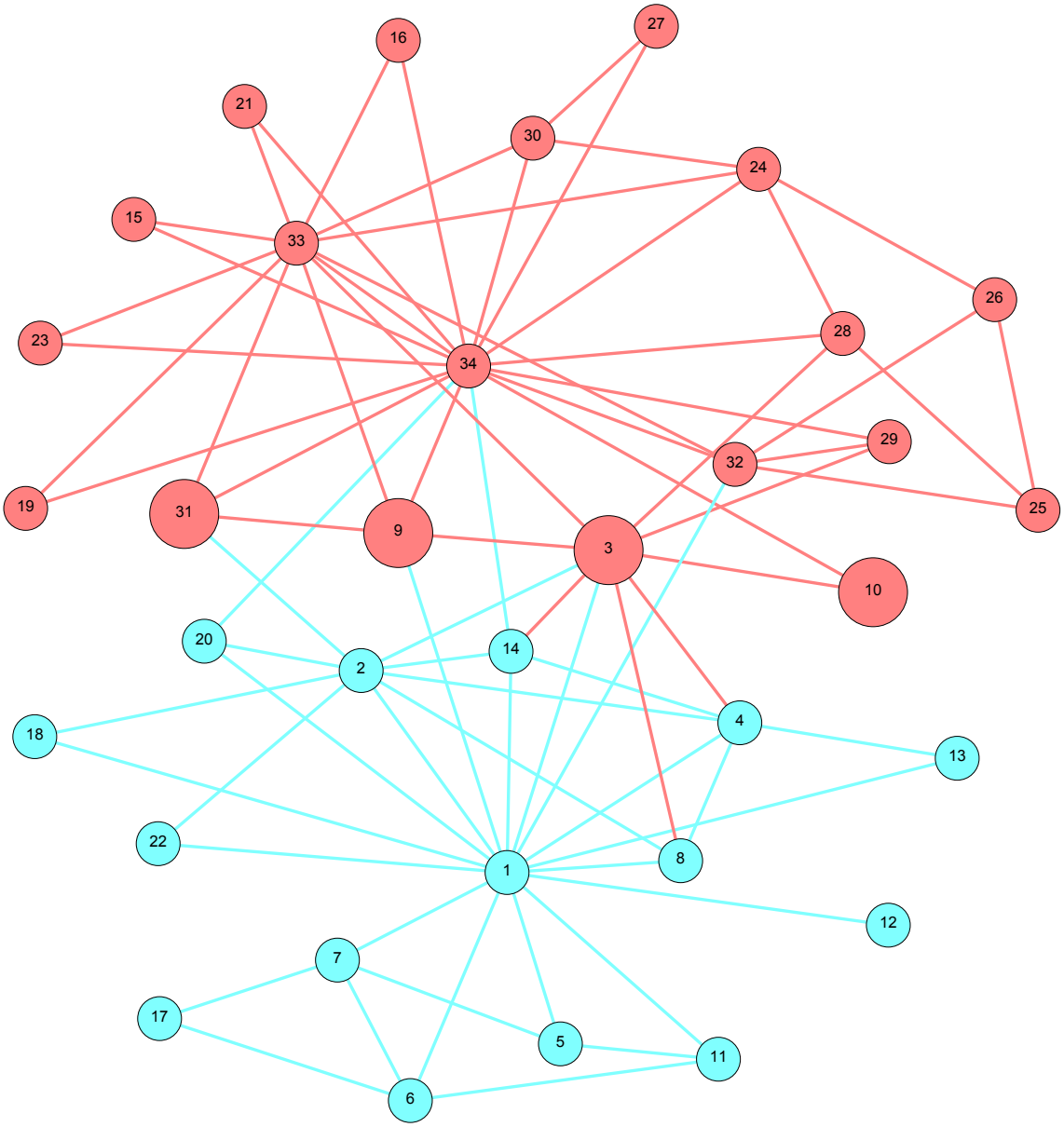
<http://snap.stanford.edu/data/amazon0312.html>

A Triedny diagram

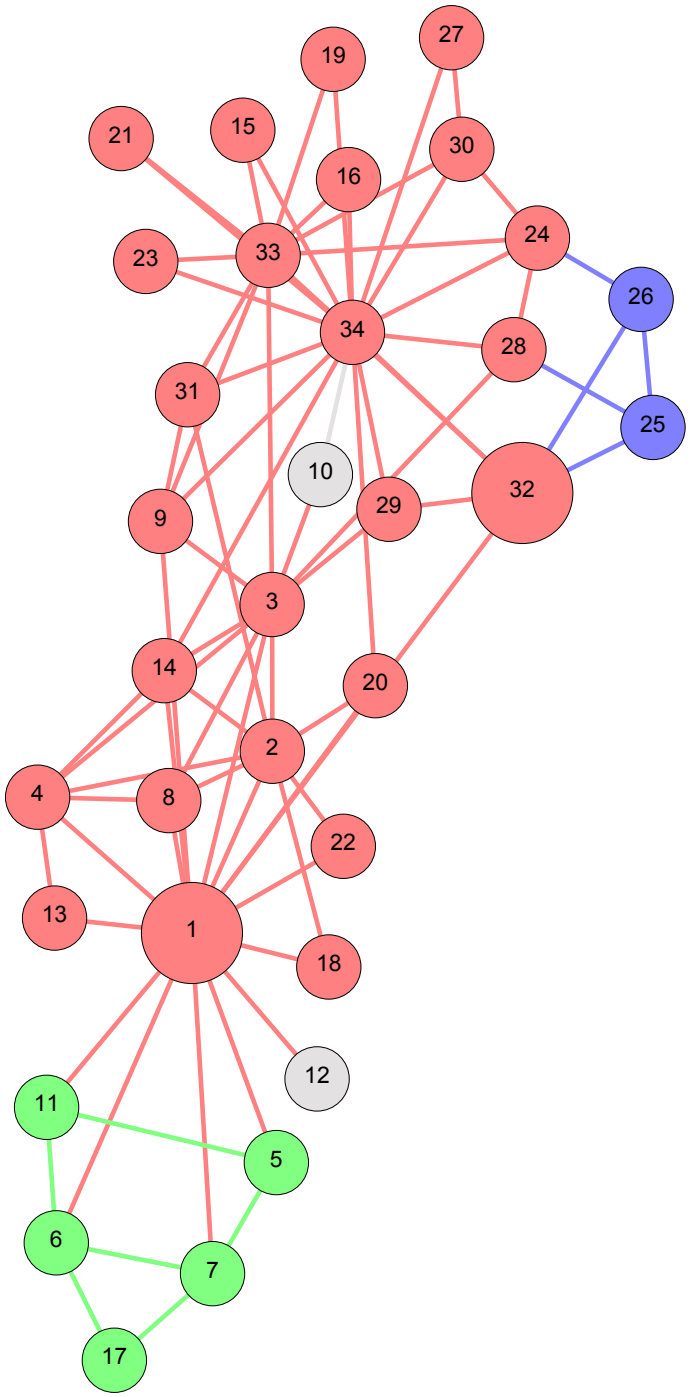


Obr. 27: Triedny diagram aplikácie

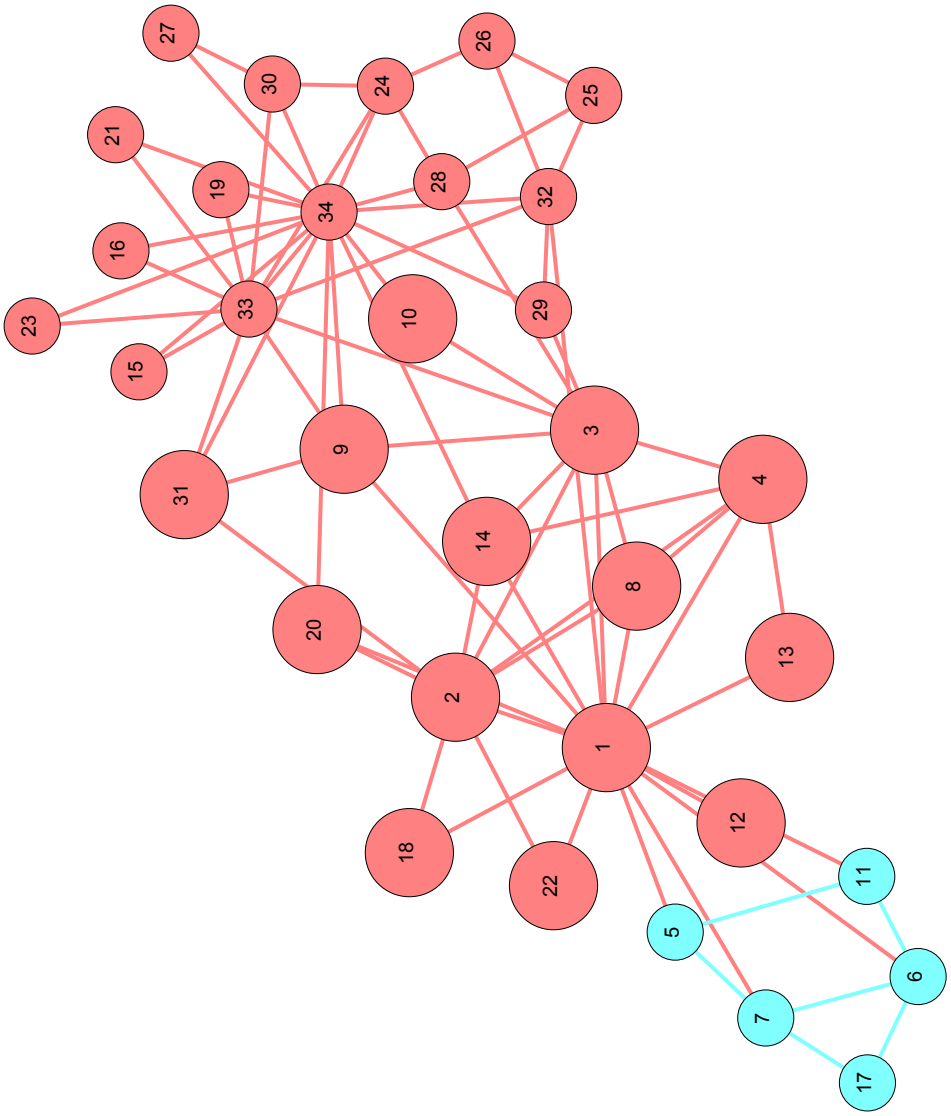
B Vizualizácie sietí



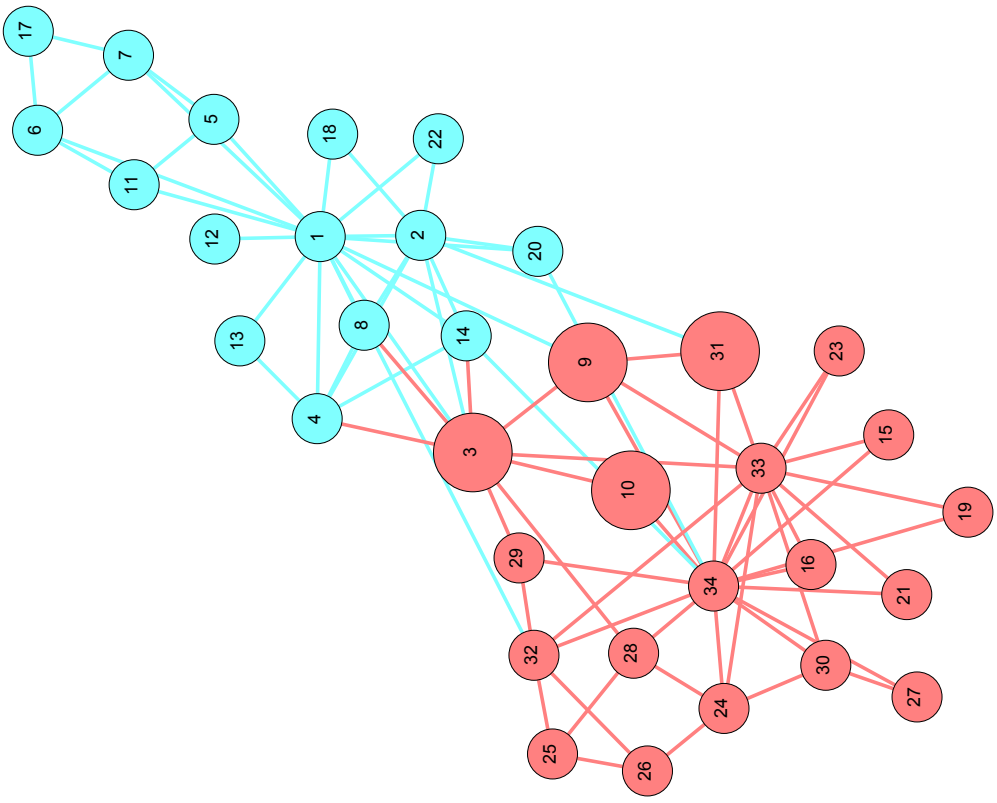
Obr. 28: Zacharyho karate klub - detekované prekrývajúce sa vrcholy.



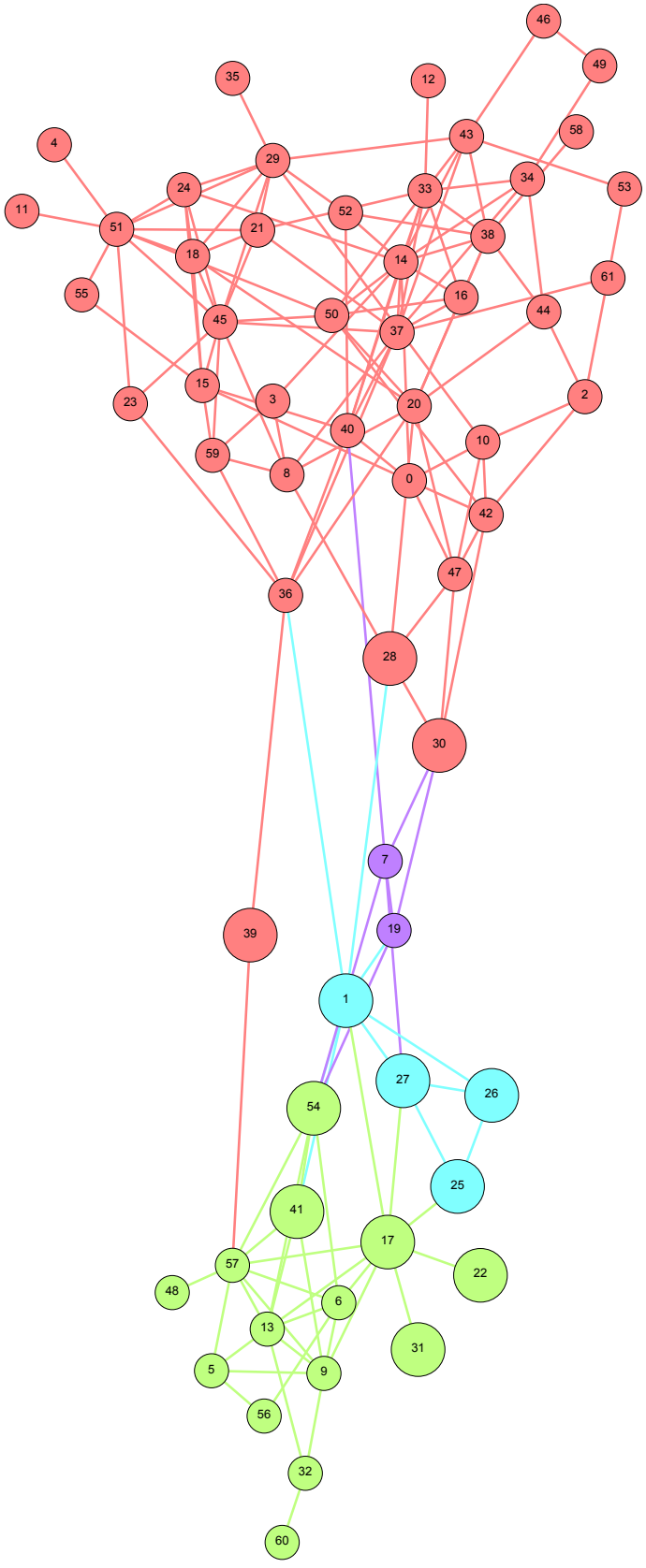
Obr. 29: Zacharyho karate klub - metóda CPM.



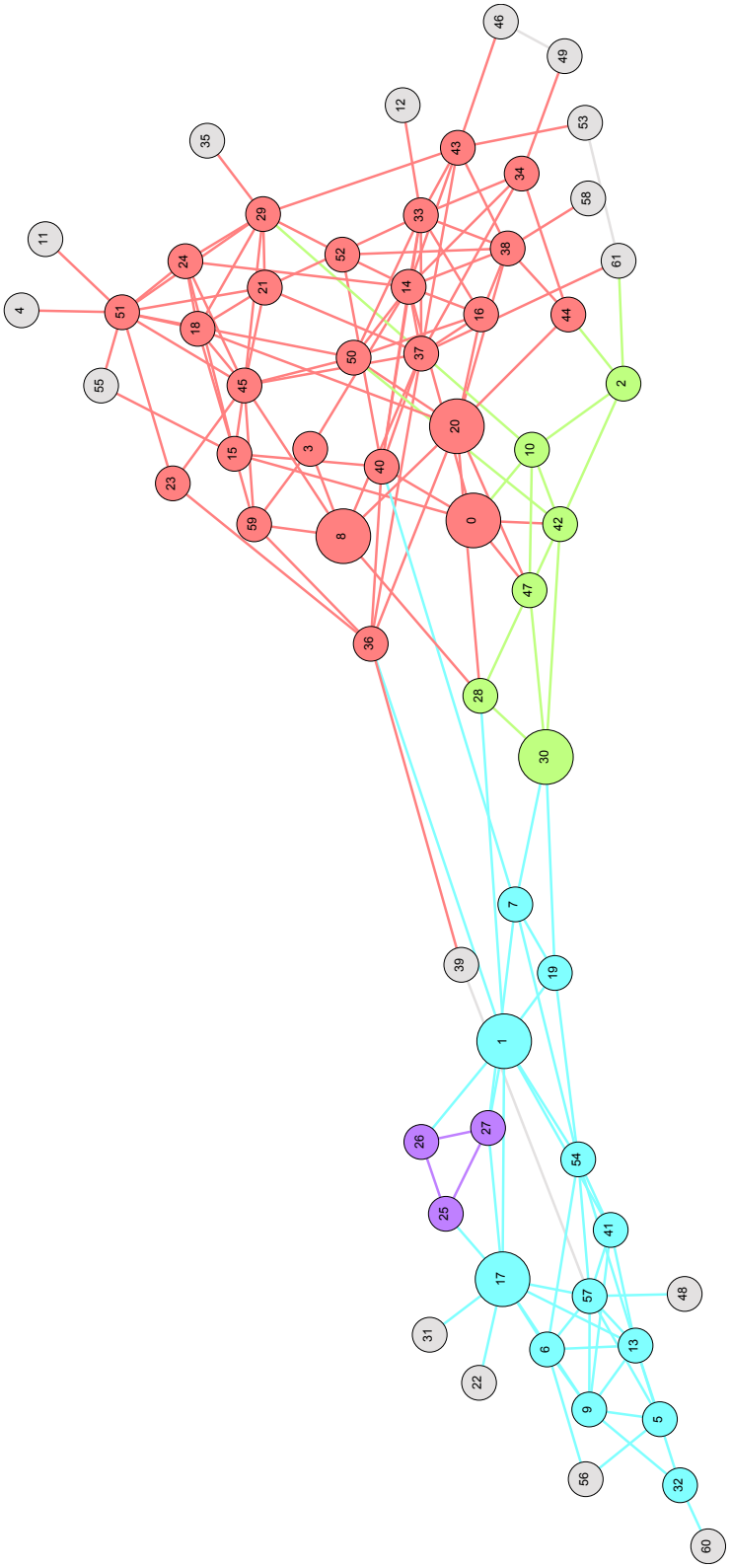
Obr. 30: Zacharyho karate klub - metóda CIS.



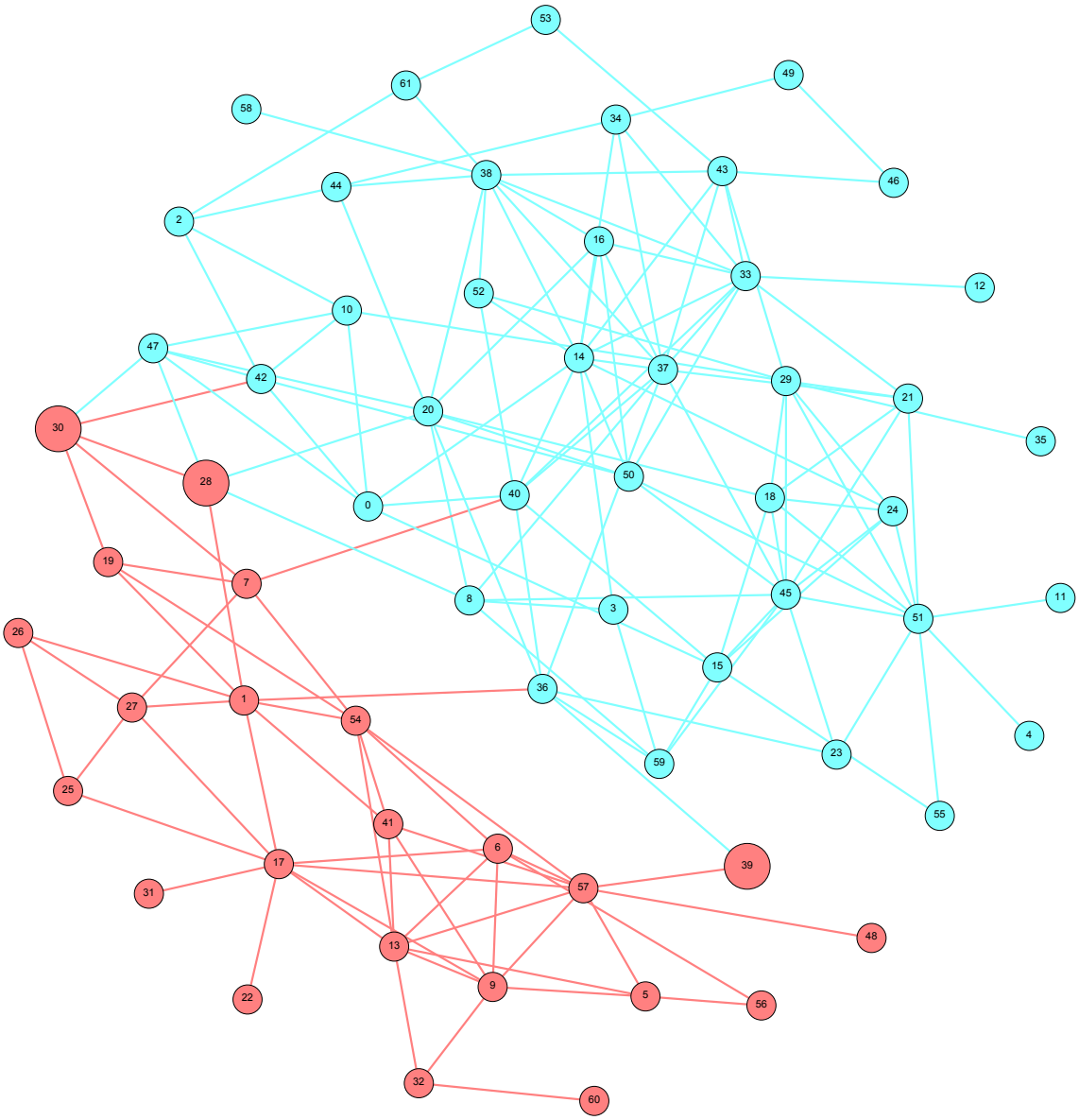
Obr. 31: Zacharyho karate klub - metóda NCIS.



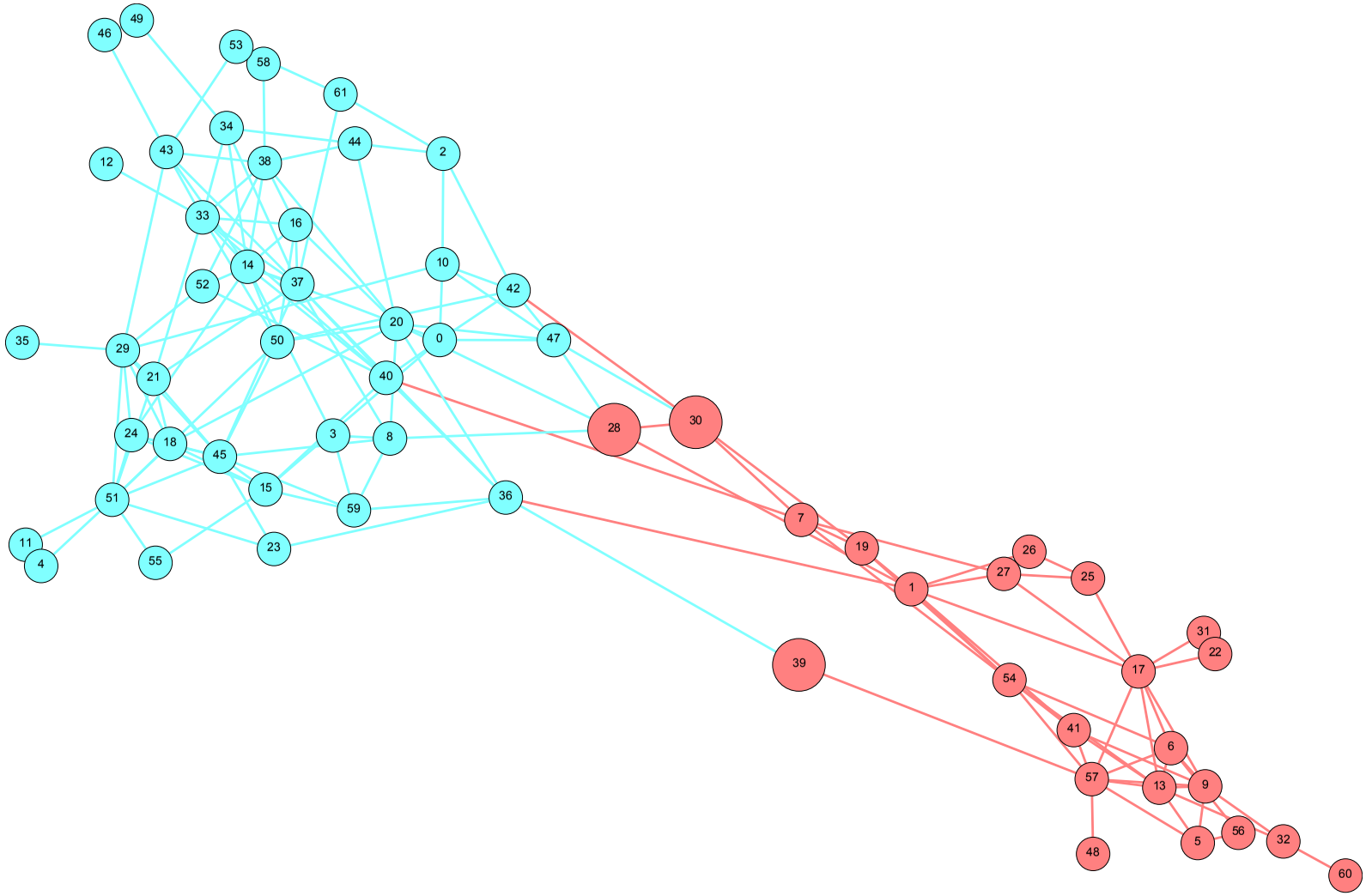
Obr. 32: Zacharyho karate klub - metóda GCIS.



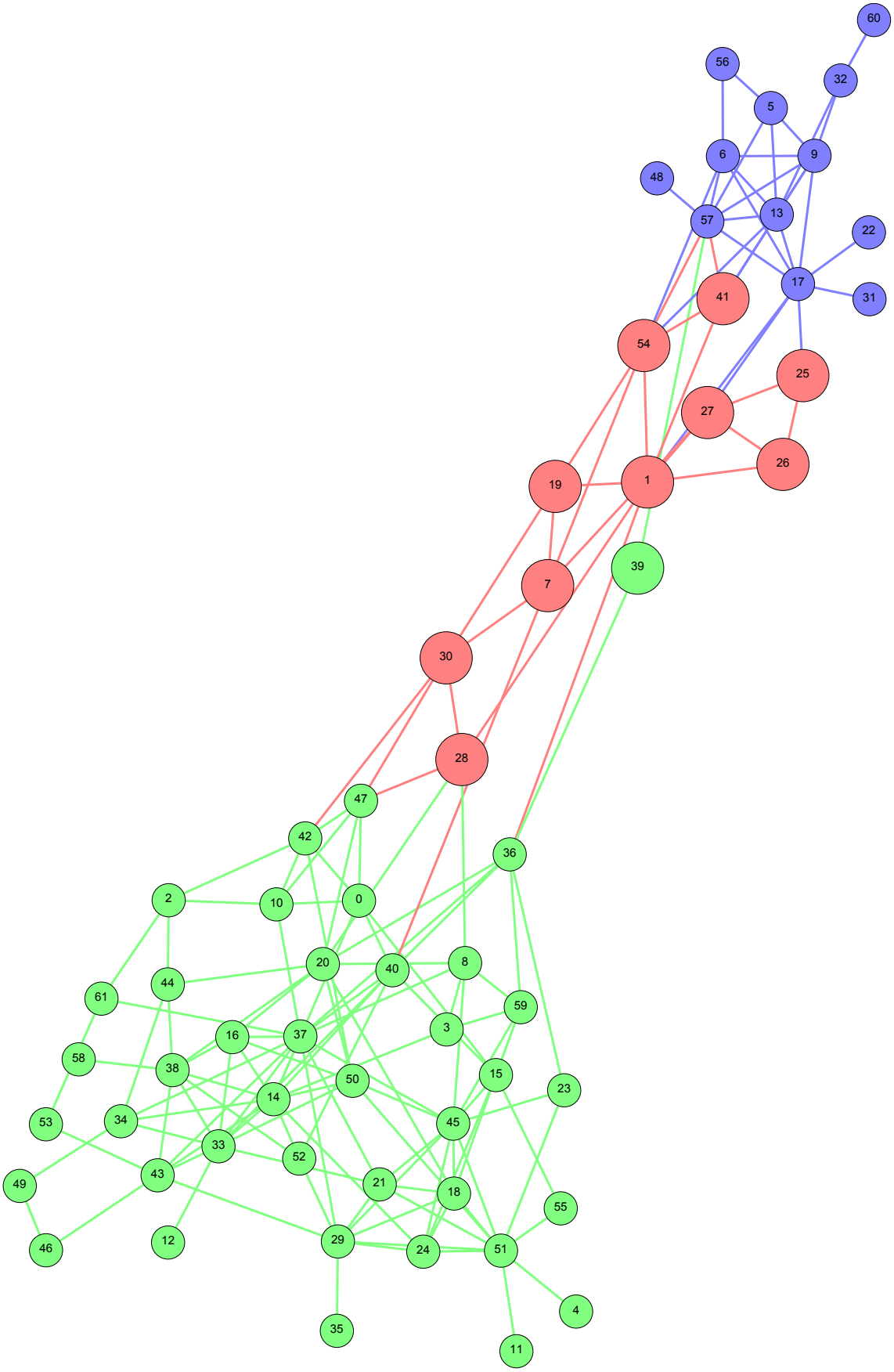
Obr. 33: Delfíni - metóda CPM.



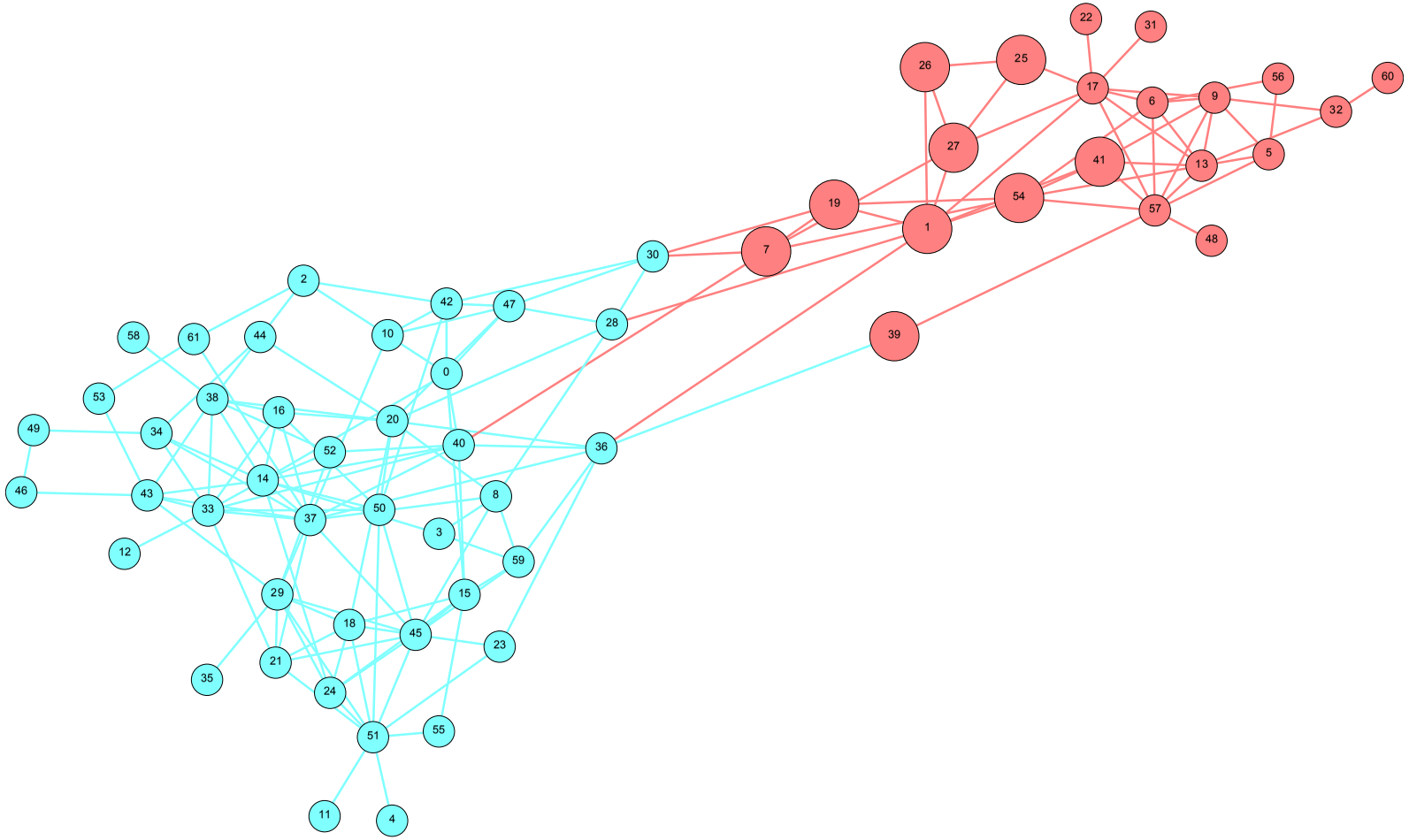
Obr. 34: Delfíni - metóda GCIS (najlepšia konfigurácia).



Obr. 35: Delfíni - metóda GCIS (najlepšia konfigurácia), č.2.

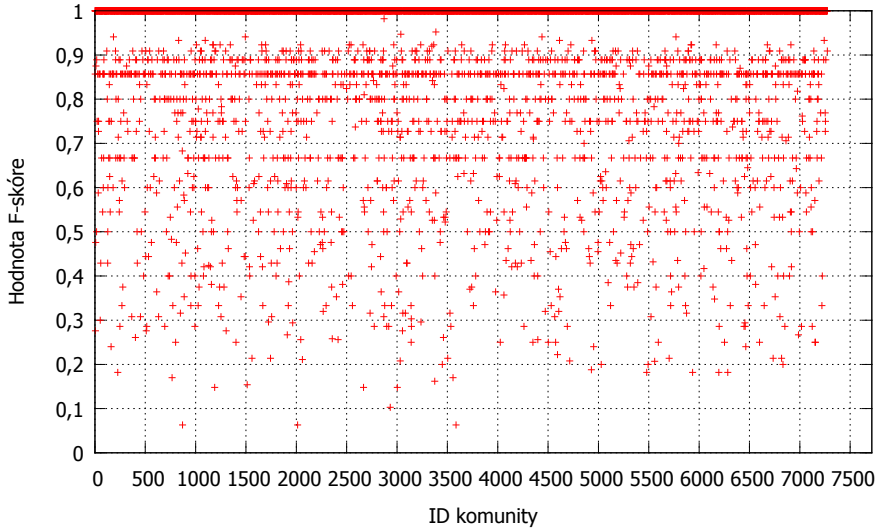


Obr. 36: Delfíni - metóda CIS.

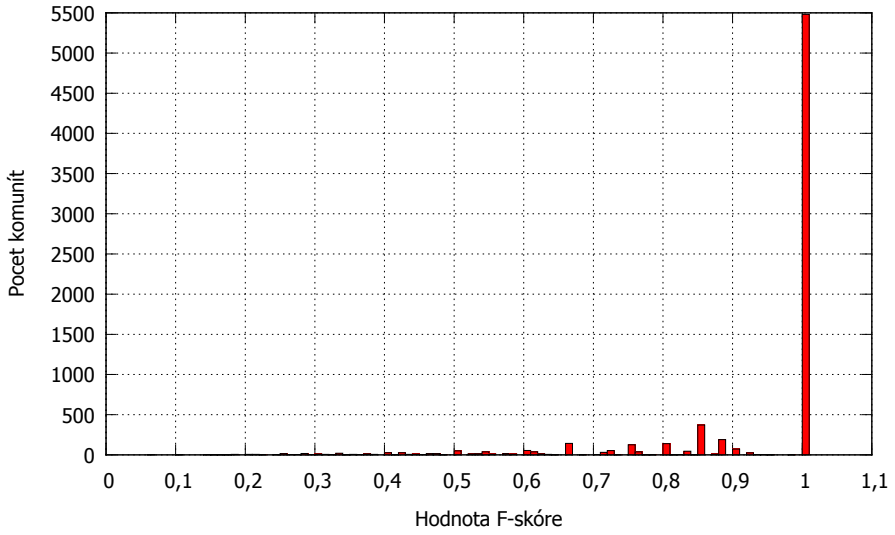


Obr. 37: Delfíni - metóda NCIS.

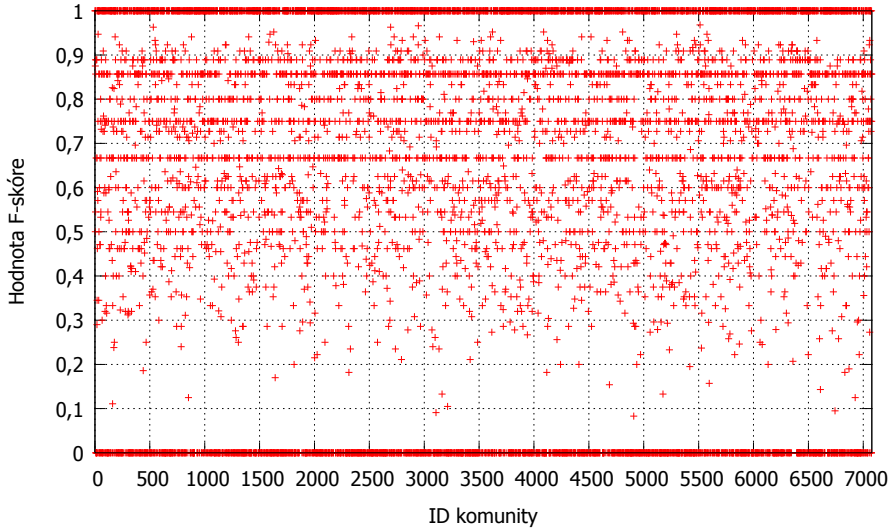
C Grafy



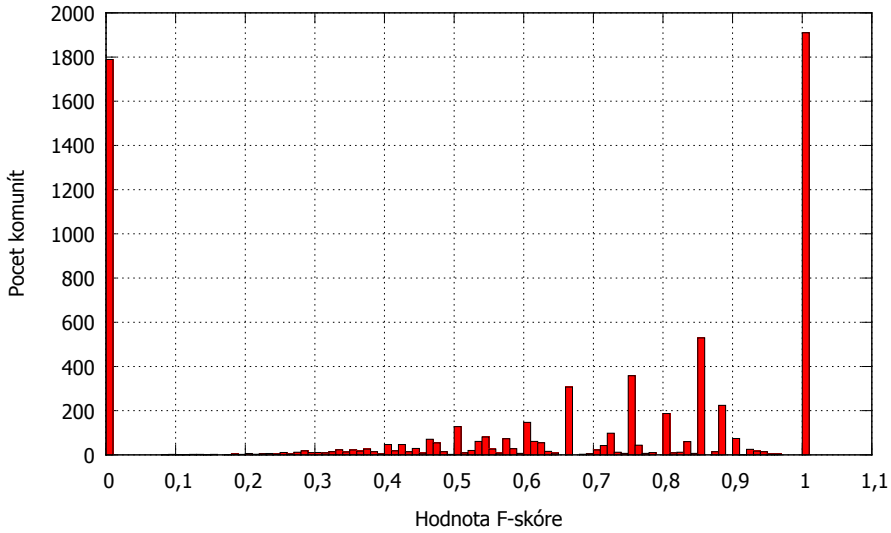
Obr. 38: Zobrazenie F-skóre vzhľadom na jednotlivé komunity - BS NCIS - DBLP



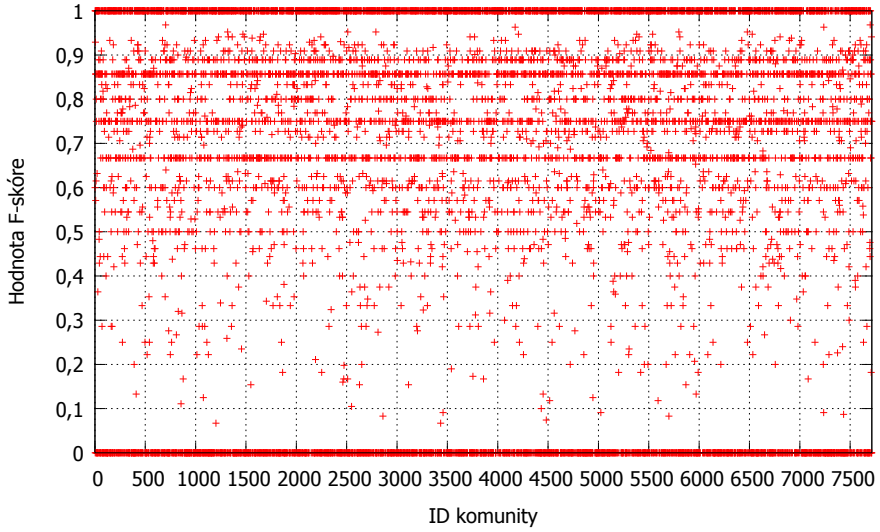
Obr. 39: Kumulatívna distribúcia F-skóre jednotlivých komunít - BS NCIS - DBLP



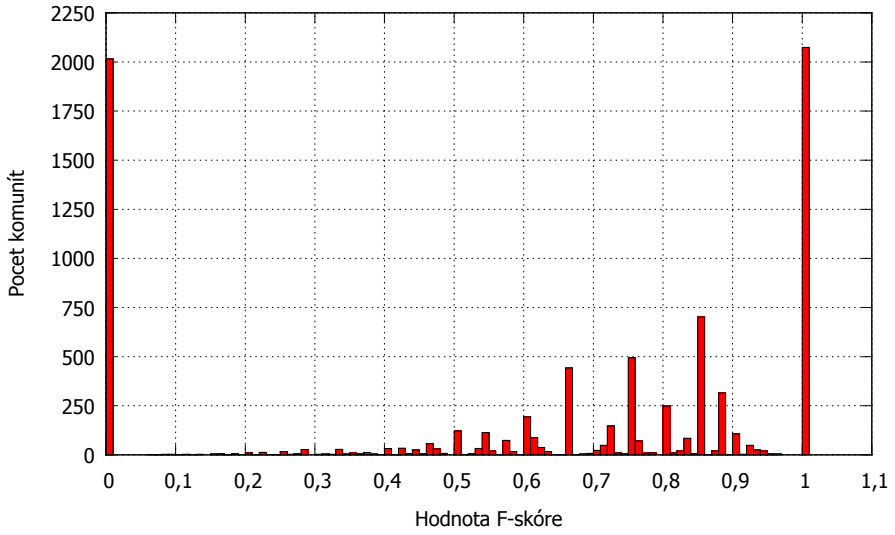
Obr. 40: Zobrazenie F-skóre vzhľadom na jednotlivé komunity - CIS - DBLP



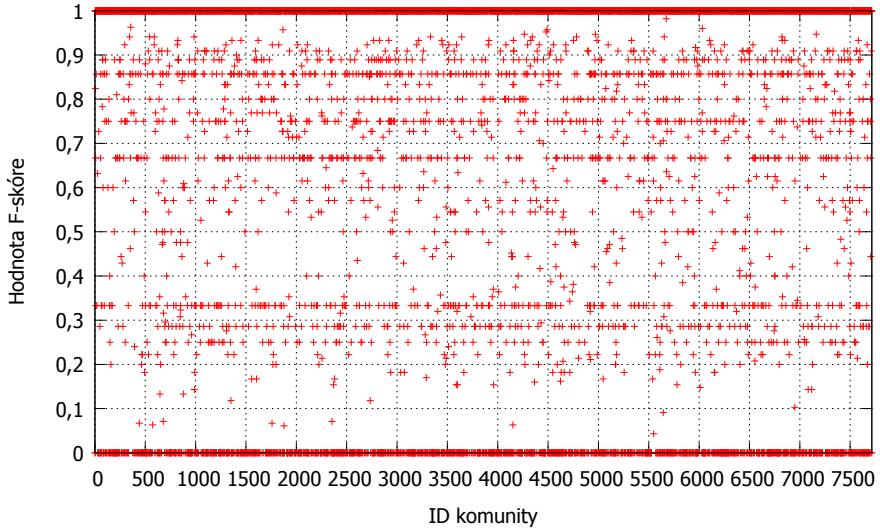
Obr. 41: Kumulatívna distribúcia F-skóre jednotlivých komunít - CIS - DBLP



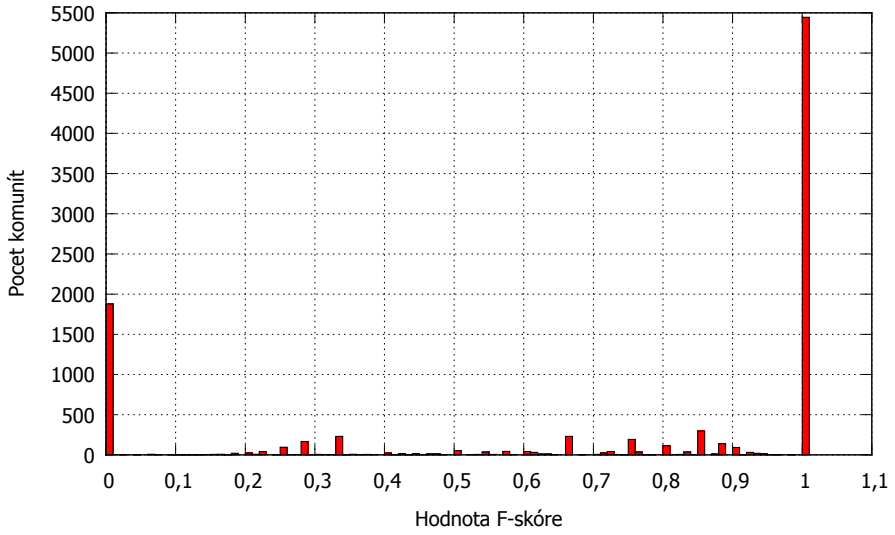
Obr. 42: Zobrazenie F-skóre vzhľadom na jednotlivé komunity - S GCIS0 - DBLP



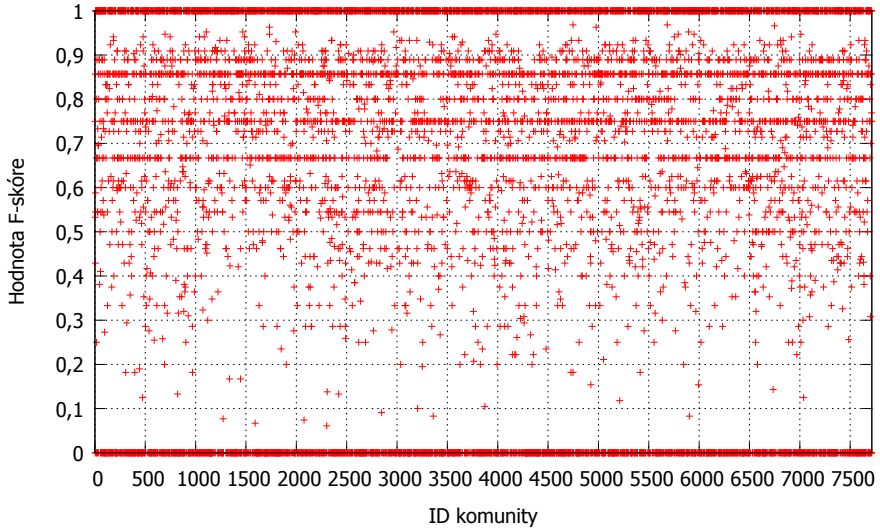
Obr. 43: Kumulatívna distribúcia F-skóre jednotlivých komunít - S GCIS0 - DBLP



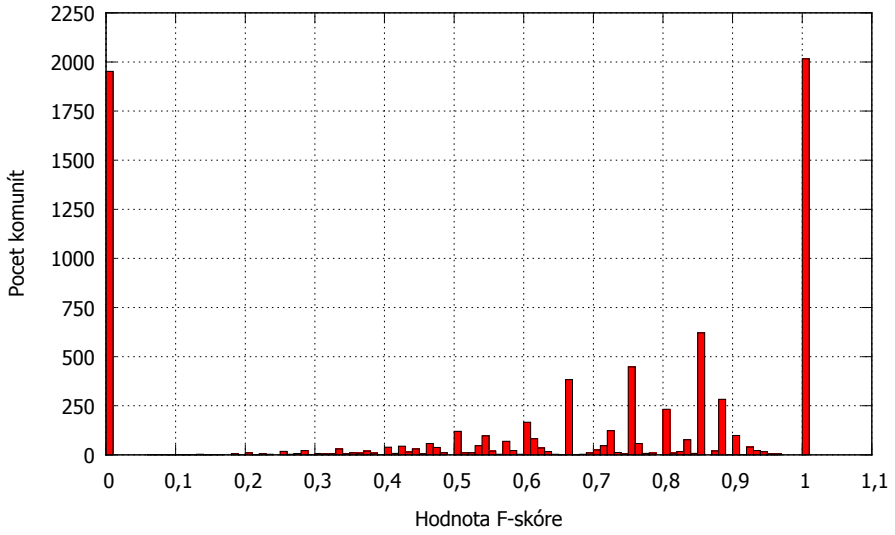
Obr. 44: Zobrazenie F-skóre vzhľadom na jednotlivé komunity - S GCIS1 - DBLP



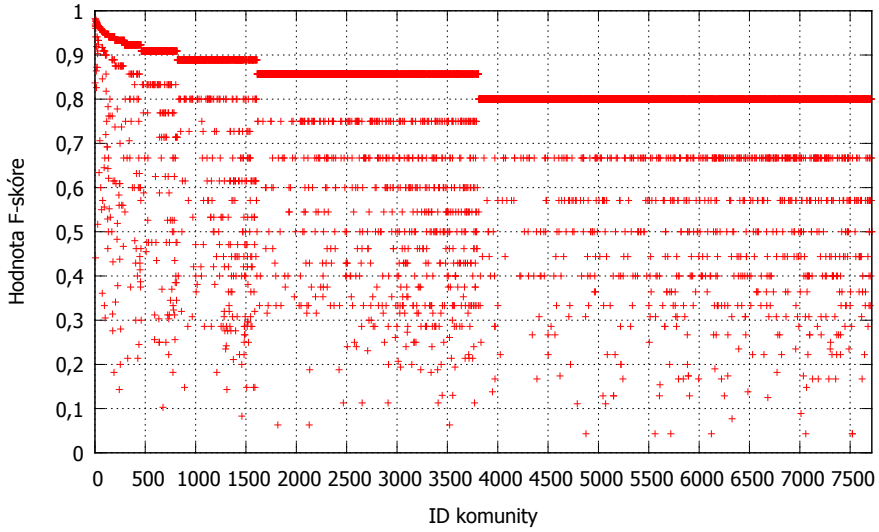
Obr. 45: Kumulatívna distribúcia F-skóre jednotlivých komunít - S GCIS1 - DBLP



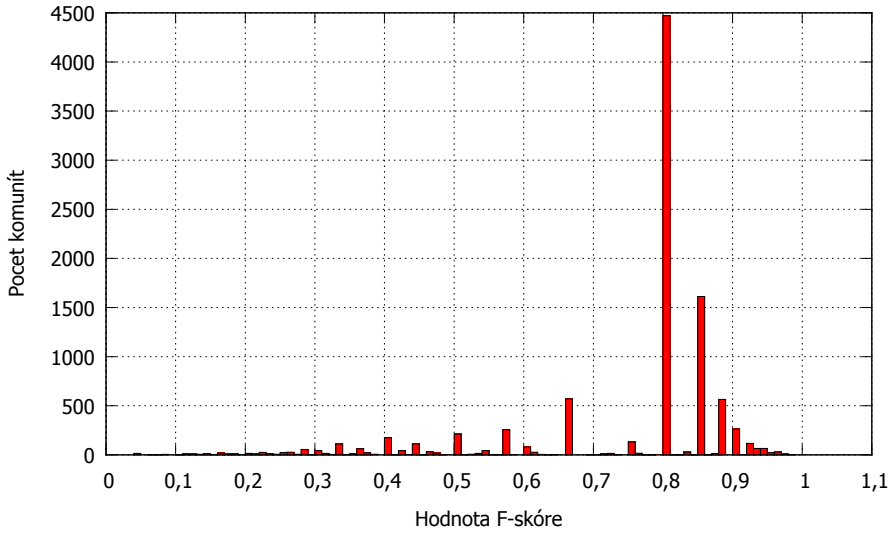
Obr. 46: Zobrazenie F-skóre vzhľadom na jednotlivé komunity - S NCIS - DBLP



Obr. 47: Kumulatívna distribúcia F-skóre jednotlivých komunít - S NCIS - DBLP



Obr. 48: Zobrazenie F-skóre vzhľadom na jednotlivé komunity - VDM - DBLP



Obr. 49: Kumulatívna distribúcia F-skóre jednotlivých komunít - VDM - DBLP

D Výpisy zdrojového kódu

```
public Graph parseFile(String pathToFile, String delimiter) {
    TObjectIntHashMap<String> addedValues = new TObjectIntHashMap<>();
    AdjacencyListGraph adjGraph = new AdjacencyListGraph();
    Mapper.idToValueMapping = new TIntObjectHashMap<String>();
    boolean wasCreatedVertexA, wasCreatedVertexB;
    BufferedReader br = null;
    try {
        br = new BufferedReader(new FileReader(pathToFile));
        String line = br.readLine();
        int vertexBID = -1;
        int vertexAID = -1;
        while (line != null) {
            wasCreatedVertexA = wasCreatedVertexB = false;
            String[] parts = line.split(delimiter);
            // ak vrchol neexistuje, tak ho vytvorime, pridame do grafu a ulozime si ho do pridanych
            if (addedValues.get(parts[0]) == 0) {
                wasCreatedVertexA = true;
                vertexAID = Mapper.getVertexId();
                Mapper.idToValueMapping.put(vertexAID, parts[0]);
                addedValues.put(parts[0], vertexAID);
                adjGraph.addVertex(vertexAID);
            }
            if (addedValues.get(parts[1]) == 0) {
                wasCreatedVertexB = true;
                vertexBID = Mapper.getVertexId();
                Mapper.idToValueMapping.put(vertexBID, parts[1]);
                addedValues.put(parts[1], vertexBID);
                adjGraph.addVertex(vertexBID);
            }
            // aktualizujeme hrany vrcholov
            if (wasCreatedVertexA && wasCreatedVertexB) {
                adjGraph.addEdge(vertexAID, vertexBID);
                adjGraph.addEdge(vertexBID, vertexAID);
            } else if (!wasCreatedVertexA && !wasCreatedVertexB) {
                adjGraph.addEdge(addedValues.get(parts[0]), addedValues.get(parts[1]));
                adjGraph.addEdge(addedValues.get(parts[1]), addedValues.get(parts[0]));
            } else if (!wasCreatedVertexA) {
                adjGraph.addEdge(vertexBID, addedValues.get(parts[0]));
                adjGraph.addEdge(addedValues.get(parts[0]), vertexBID);
            } else {
                adjGraph.addEdge(vertexAID, addedValues.get(parts[1]));
                adjGraph.addEdge(addedValues.get(parts[1]), vertexAID);
            }
            line = br.readLine();
        }
    } ... spracovanie vynimiek
    addedValues = null;
    adjGraph.sortNeigborsLists();
    return adjGraph;
```

Výpis 7: Spracovanie súboru s grafom

```
public static void exportToGexf(String dirPath, Graph g, Collection<TIntList> communities) {
    Gexf gexf = new GexfImpl();
    Calendar date = Calendar.getInstance();
    gexf.getMetadata().setLastModified(date.getTime()).setCreator("CommunityFinder");
    gexf.setVisualization(true);
    it.uniroma1.dis.wsngroup.gexf4j.core.Graph graph = gexf.getGraph();
    graph.setDefaultEdgeType(EdgeType.UNDIRECTED).setMode(Mode.STATIC);
    AttributeList attrList = new AttributeListImpl(AttributeClass.NODE);
    graph.getAttributeLists().add(attrList);
    Attribute attCommunity = attrList.createAttribute(AttributeType.STRING, "community");
    List<Color> colorList = generateColors(communities.size());
    // vytvorime vrcholy
    TIntObjectHashMap<Node> idToNodes = new TIntObjectHashMap<Node>(g.
        getVerticesCount());
    TIntHashSet wasProcessed = new TIntHashSet(g.getVerticesCount());
    int[] allVertices = g.getAllVertices();
    for (int i = 0; i < allVertices.length; i++) {
        int vertex = allVertices[i];
        Node actualNode;
        if (!idToNodes.contains(vertex)) {
            actualNode = graph.createNode(String.valueOf(vertex));
            actualNode.setLabel(String.valueOf(Mapper.idToValueMapping.get(vertex))).setSize(
                NORMAL_SIZE).setColor(new MyColor(227, 225, 225)).getAttributeValues().
                addValue(attCommunity, NO_COMMUNITY_ID_STRING);
            idToNodes.put(vertex, actualNode);
        } else
            actualNode = idToNodes.get(vertex);
        // vytvorime susedov
        TIntList neighbors = g.getVertexNeighbors(vertex);
        for (int j = 0; j < neighbors.size(); j++) {
            int v = neighbors.get(j);
            if (wasProcessed.contains(v))
                continue;
            // ak koncovy vrchol este nebol vytvoreny
            Node newNode;
            if ((newNode = idToNodes.get(v)) == null) {
                newNode = graph.createNode(String.valueOf(v));
                newNode.setLabel(String.valueOf(Mapper.idToValueMapping.get(v))).setSize(
                    NORMAL_SIZE).setColor(new MyColor(227, 225, 225)).getAttributeValues().
                    addValue(attCommunity, NO_COMMUNITY_ID_STRING);
                idToNodes.put(v, newNode);
            }
            actualNode.connectTo(newNode);
        }
        wasProcessed.add(vertex);
    }
    // zaradenie vrcholov na zaklade komunit
    wasProcessed.clear();
    int communityIdCounter = 1;
    StringBuilder commsAttSb = new StringBuilder();
    for (TIntList community : communities) {
        for (int i = 0; i < community.size(); i++) {
            int vertex = community.get(i);
```

```
                if (wasProcessed.contains(vertex)) {
                    Node overlapNode = idToNodes.get(vertex);
                    overlapNode.setSize(OVERLAP_SIZE);
                    overlapNode.getShapeEntity().setNodeShape(NodeShape.DIAMOND);
                    commsAttSb.append(overlapNode.getAttributeValues().get(
                            COMMUNITY_ATTR_POSITION).getValue()).append(",").append(
                            communityIdCounter);
                    overlapNode.getAttributeValues().clear();
                    overlapNode.getAttributeValues().addValue(attCommunity, commsAttSb.toString());
                    commsAttSb.setLength(0);
                } else {
                    Color c = colorList.get(communityIdCounter - 1);
                    Node normalNode = idToNodes.get(vertex);
                    normalNode.getAttributeValues().clear();
                    normalNode.setColor(new MyColor(c.getRed(), c.getGreen(), c.getBlue())).
                            getAttributeValues().addValue(attCommunity, String.valueOf(
                            communityIdCounter));
                    wasProcessed.add(vertex);
                }
            }
            communityIdCounter++;
        }
        ... samotny zapis suboru
    }
```

Výpis 8: Vygenerovanie gexf súboru

```
public ArrayList<TIntList> findMaximalCliques(Graph graph) {
    maximalCliques = new ArrayList<TIntList>();
    // v popise algoritmu mnozina S
    TIntList not = new TIntArrayList();
    // v popise algoritmu mnozina R
    TIntList compSub = new TIntArrayList();
    // v popise algoritmu mnozina K
    TIntList candidates;
    // inicializacia kandidatov
    sortedVertices = new TIntArrayList(graph.getAllVertices());
    Sorter.sort(sortedVertices, graph, SortOrder.DESCENDING);
    candidates = new TIntArrayList(sortedVertices);
    bronKerboschGraph = graph;
    bronKerbosch(compSub, candidates, not);
    return maximalCliques;
}
// BK - algoritmus
private void bronKerbosch(TIntList compSub, TIntList candidates, TIntList not) {
    if (candidates.isEmpty() && not.isEmpty()) {
        maximalCliques.add(new TIntArrayList(compSub));
    } else if (!candidates.isEmpty()) {
        pivot = NO_VERTEX;
        for (int i = 0; i < sortedVertices.size(); i++) {
            int v = sortedVertices.get(i);
            if (pivot == NO_VERTEX) {
                pivot = v;
            }
```

```
            if (candidates.contains(v) || not.contains(v)) {
                pivot = v;
                break;
            }
        }
        // prejdeme kandidatov a vyberieme vhodne vrcholy na spracovanie
        TIntList helperList = new TIntArrayList();
        for (int i = 0; i < candidates.size(); i++) {
            if (!bronKerboschGraph.getVertexNeighbors(candidates.get(i)).contains(pivot))
                helperList.add(candidates.get(i));
        }
        for (int i = 0; i < helperList.size(); i++) {
            int candidate = helperList.get(i);
            TIntList candidateNeighbors = bronKerboschGraph.getVertexNeighbors(candidate);
            TIntList newCandidates = retainAllCustom(candidates, candidateNeighbors);
            TIntList newCompSub = new TIntArrayList(compSub);
            newCompSub.add(candidate);
            TIntList newNot = retainAllCustom(not, candidateNeighbors);
            if (newCandidates.isEmpty() && newNot.isEmpty()) {
                maximalCliques.add(new TIntArrayList(newCompSub));
            } else {
                // rekurzia
                bronKerbosch(newCompSub, newCandidates, newNot);
            }
            candidates.remove(candidate);
            not.add(candidate);
        }
    }
}
```

Výpis 9: Implementácia Bron-Kerbosch algoritmu

```
private Collection<TIntList> findCommunities(TIntByteHashMap cliqueOverlapMatrix, int k){
    Collection<TIntList> communities = new LinkedHashSet<TIntList>();
    // reprezentuje kolekciu komunit (zlucenych klik) kazdeho riadku. Pracujeme len s indexami klik
        .
    Collection<TIntList> rowCommunities = new ArrayList<TIntList>();
    for (int row = 0; row < maxCliqueSize; row++){
        // do listu zlucime indexy jednotlivych klik na riadku - riadkova komunita
        TIntList rowCliqueIndexes = new TIntArrayList();
        for (int col = 0; col < maxCliqueSize-row; col++) {
            if (cliqueOverlapMatrix.get(row + col * hashMultiplier) == OVERLAP_IS_SUFFICIENT){
                // pridavame kliku s poziciou col+row, lebo pracujeme so "skosenou" maticou
                rowCliqueIndexes.add(col+row);
            }
            // potrebujeme zahrnut aj symetricke prvky
            if (row > 0 && col < row)
                if (cliqueOverlapMatrix.get(col + (row-col) * hashMultiplier) ==
                        OVERLAP_IS_SUFFICIENT)
                    rowCliqueIndexes.add(col);
        }
        if (rowCommunities.isEmpty()) {
            if (!rowCliqueIndexes.isEmpty())
                rowCommunities.add(rowCliqueIndexes);
```

```
            }
            else if (!rowCliqueIndexes.isEmpty()){
                Collection<TIntList> mergedWith = new ArrayList<TIntList>();
                boolean unionWasMade = false;
                for (TIntList rowCommunity : rowCommunities){
                    if (retainAllCustom(rowCliqueIndexes, rowCommunity).size() > 0){
                        unionWasMade = true;
                        // spravime zjednotenie novej riadkovej komunity s aktualne iterovanou
                        addAllCustom(rowCliqueIndexes, rowCommunity);
                        // ulozime aktualne iterovanu riadkovu komunitu z "rowCommunities" k odstraneniu
                        mergedWith.add(rowCommunity);
                    }
                }
                // potrebujeme odstranit stare riadkove komunity, ktore mali zaniknut pri zluceni a do "
                    rowCommunities"
                // pridat novo zlucenu komunitu
                if (unionWasMade){
                    for (TIntList list : mergedWith) {
                        rowCommunities.remove(list);
                    }
                }
                rowCommunities.add(rowCliqueIndexes);
            }
        }
        // vytvorenie realnych komunit z riadkovych komunit
        int cliquePosition;
        for (TIntList currentRowCommunity : rowCommunities) {
            TIntList tempCommunity = new TIntArrayList();
            for (int i = 0; i < currentRowCommunity.size(); i++) {
                cliquePosition = currentRowCommunity.get(i);
                addAllCustom(tempCommunity, lastMaximalCliques.get(cliquePosition));
            }
            communities.add(tempCommunity);
        }
        return communities;
    }
```

Výpis 10: Zlúčenie prekrývajúcich sa klík do komunít

```
    protected void postProcessFilter(Collection<TIntList> communities, int[] allVertices ) {
        if (communities.size() > 1) {
            for (TIntList community1 : communities) {
                if (community1.size() == allVertices.length) {
                    communities.remove(community1);
                    break;
                }
            }
        }
        TIntList [] communitiesList = communities.toArray(new TIntList[0]);
        TIntHashSet removedIndexes = new TIntHashSet();
        for (int i = 0; i < communitiesList.length; i++) {
            if (removedIndexes.contains(i))
                continue;
            for (int j = i + 1; j < communitiesList.length; j++) {
```

```
            if (isContainedInCommunity(communitiesList[j], communitiesList[i])) {
                communities.remove(communitiesList[i]);
                break;
            } else if (isContainedInCommunity(communitiesList[i], communitiesList[j])) {
                communities.remove(communitiesList[j]);
                removedIndexes.add(j);
            }
        }
    }
    // zlucenie podobnych komunit
    boolean shouldMerge = true;
    while (shouldMerge) {
        TIntIntHashMap mergedCommIDs = new TIntIntHashMap(communitiesList.length / 2);
        communitiesList = communities.toArray(new TIntList[0]);
        for (int i = 0; i < communitiesList.length; i++) {
            if (i != 0 && mergedCommIDs.get(i) != mergedCommIDs.getNoEntryValue())
                continue;
            double similarityMeasure = 0;
            double bestSimilarityMeasure = 0;
            int commIDtoMerge = -1;
            for (int j = i + 1; j < communitiesList.length; j++) {
                if (mergedCommIDs.get(j) != mergedCommIDs.getNoEntryValue())
                    continue;
                similarityMeasure = calculateSimilarityMeasure(communitiesList[i], communitiesList[j])
                        ;
                if (similarityMeasure > bestSimilarityMeasure) {
                    bestSimilarityMeasure = similarityMeasure;
                    commIDtoMerge = j;
                }
            }
            if (bestSimilarityMeasure * 100 > SIMILARITY_THRESHOLD) {
                mergedCommIDs.put(commIDtoMerge, i);
            }
        }
        // zlucenie
        for ( TIntIntIterator it = mergedCommIDs.iterator(); it.hasNext();) {
            it.advance();
            TIntList community1 = communitiesList[it.key()];
            TIntList community2 = communitiesList[it.value()];
            communities.remove(community2);
            communities.remove(community1);
            communities.add(unionAllCustom(community1, community2));
        }
        if (mergedCommIDs.size() > 0)
            shouldMerge = true;
        else
            shouldMerge = false;
    }
}
```

Výpis 11: Post processing komunít u metód založených na CIS

```
...
TIntList communityVerticesAndNeighb = getCommunityVerticesAndNeigbors(comm);
```

```
bestLocalDensity = −1;
bestLocalVertex = −1;
// najdeme vertex, ktoreho pridanim sa najviac zvacsi density setu
for (int it = 0; it < communityVerticesAndNeighb.size(); it++) {
    comparedVertex = communityVerticesAndNeighb.get(it);
    if (comm.binarySearch(comparedVertex) < 0) {
        comm.add(comparedVertex);
        comm.sort();
        newDensity = calculateDensity(comm, tuningParam);
        if (!( newDensity > oldDensity))
            comm.remove(comparedVertex);
        else if (newDensity > bestLocalDensity) {
            bestLocalDensity = newDensity;
            bestLocalVertex = comparedVertex;
        }
        comm.remove(comparedVertex);
    }
}
// pridame najvhodnejsi uzol − ten, ktory najviac zvysi density setu − ak existuje
if (bestLocalVertex != −1) {
    comm.add(bestLocalVertex);
    oldDensity = bestLocalDensity;
    improved = true;
}
//odobranie vsetkych nevhodnych
//musime aktualizovat susedov
communityVerticesAndNeighb = getCommunityVerticesAndNeigbors(comm);
for (int it = 0; it < communityVerticesAndNeighb.size(); it++) {
    comparedVertex = communityVerticesAndNeighb.get(it);
    if (comm.size() > 2 && comm.remove(comparedVertex)) {
        newDensity = calculateDensity(comm, tuningParam);
        if (!( newDensity > oldDensity)) {
            comm.add(comparedVertex);
            comm.sort();
        } else {
            oldDensity = newDensity;
            improved = true;
        }
    }
}
...
```

Výpis 12: Priebeh skenu v metóde GCIS

E Obsah priloženého CD

Názov zložky - popis

- *Aplikacia* - spustitelná aplikácia v podobe .jar súboru
- *Projekt* - zdrojové kódy aplikácie vo forme Eclipse projektu
- *Experimenty* - súbory, ktoré vznikli pri experimentoch. Zahŕňajú .svg obrázky sietí, súbory s nájdenými komunitami a súbory s ostatnými informáciami pre jednotlivé metódy použité pri testovaní
- *Testovacie data* - .csv a .txt súbory s testovacími dátami
- *Obrazky* - vizualizované siete, doplňujúce grafy a triedny diagram
- *Text* - text práce vo formáte .pdf